Lecture Notes in Computer Science 5515

Jürgen Lerner   Dorothea Wagner
Katharina A. Zweig (Eds.)

Algorithmics of Large and Complex Networks

Design, Analysis, and Simulation

Springer

Volume Editors

Jürgen Lerner
University of Konstanz, Department of Computer & Information Science
78457 Konstanz, Germany
E-mail: lerner@inf.uni-konstanz.de

Dorothea Wagner
Technical University of Karlsruhe, Institute of Theoretical Informatics
76128 Karlsruhe, Germany
E-mail: wagner@ira.uka.de

Katharina A. Zweig
Eötvös Lorand University, Department of Biological Physics
Pazmany P. Stny 1A, 1117 Budapest, Hungary
E-mail: nina@ninasnet.de

Library of Congress Control Number: 2009929640

CR Subject Classification (1998): F.2, C.2, G.2-3, I.3.5, G.1.6, K.4.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-02093-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-02093-3 Springer Berlin Heidelberg New York

springer.com

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12679300 06/3180 5 4 3 2 1 0

Preface

Networks play a central role today, since many aspects of information technology like communication, mobility, and transport are based on networks. Social interactions and political activities of single persons as well as large organizations also rely on networks. In these times of globalization, our social and economic networks have certainly become ever larger and more complex and at the same time the need to understand their structure and the processes on top of them has become more and more important for every single country. This fact was demonstrated recently in the current global financial crisis with its complex and nearly incomprehensible entanglements of various structures and its huge effect on seemingly unrelated institutions and organizations. With higher automatization and larger computing power, our knowledge about biological networks, e.g., gene expression data or metabolic networks, has also strongly increased due to the large amount of data available. Finally, the Internet which can itself be seen as a large and complex network, has enabled us to build on top of it various kinds of networks. Whereas the latter's structure can be influenced by the software that mediates its building, the structure of our social and economic or existing biological networks is not easily controllable but at least analyzable. In all of these cases there is an intricate interplay between the structure of the network and processes on top of it or problems that have to be solved on it. Thus, it is on the one hand necessary to design efficient algorithms to study the structure of a given network or—if the structure can be controlled—to find out what would be the best network structure for a given problem. On the other hand, it is neccessary to design new and more efficient algorithms for solving various problems on these networks since many of them have become so large and complex that classical algorithms are no longer sufficient.

This volume surveys the progress in selected aspects of this important and growing field. It emerged from a research program funded by the German Research Foundation (DFG) as priority program 1126 on *Algorithmics of Large and Complex Networks*. The program consisted of projects focusing on the design of new discrete algorithms for large and complex networks. The aim of this priority program was to bring together scientists that work on various aspects of this field, to promote and support algorithmic research in this realm, and to enable new applications on the basis of the resulting algorithms. In its course, basic algorithmic methods have been developed further with respect to current requirements from applications, and new network design and network analysis tools have been proposed.

The research program was prepared collaboratively by Martin Dietzfelbinger, Michael Kaufmann, Ernst W. Mayr, Friedhelm Meyer auf der Heide, Rolf H. Möhring, Dorothea Wagner (coordinator), and Ingo Wegener. The first meetings took place in 1999 in Berlin and Bonn. After a short version was submitted to the DFG in December 1999, a grant proposal was worked out during a workshop at Schloss Dagstuhl in February 2000, submitted to the DFG on March 10, and the program was granted in the spring meeting of the DFG Senat. The duration of the program was six years, divided into three periods of two years each. Altogether, 24 projects have been sponsored out of which 11 projects ranged over all three periods, five over two periods and eight projects received founding for one period.

The chapters of this volume summarize results of projects realized within the program and survey-related work. The volume is divided into four parts: *network algorithms*, *traffic networks*, *communication networks*, and *network analysis and simulation.*

The first part presents research on general algorithmic problems that frequently occur in the context of large and complex networks. More specifically:

- In “Design and Engineering of External Memory Traversal Algorithms for General Graphs” Deepak Ajwani and Ulrich Meyer survey the state of the art in I/O-efficient graph traversal algorithms.
- In “Minimum Cycle Bases and Their Applications” Franziska Berger, Peter Gritzmann, and Sven de Vries survey polynomial-time algorithms for constructing minimum bases of the cycle space of graphs and digraphs.
- In A Survey on Approximation Algorithms for Scheduling with Machine Unavailability” Florian Diedrich, Klaus Jansen, Ulrich M. Schwarz, and Denis Trystram present recent contributions to sequential job scheduling with online and offline models of machine unavailability.
- In “Iterative Compression for Exactly Solving NP-Hard Minimization Problems” Jiong Guo, Hannes Moser, and Rolf Niedermeier survey the conceptual framework for developing fixed-parameter algorithms for NP-hard minimization problems using the iterative compression technique from Reed, Smith, and Vetta (2004).
- In “Approaches to the Steiner Problem in Networks” Tobias Polzin and Siavash Vahdati-Daneshmand overview relaxation algorithms, heuristics, and reduction approaches to the Steiner problem and show how these components can be integrated into an exact algorithm.
- In “A Survey on Multiple Objective Minimum Spanning Tree Problems” Stefan Ruzika and Horst W. Hamacher review the literature on minimum spanning tree problems with two or more objective functions each of which is of the sum or bottleneck type.

The second part deals with algorithms for traffic networks, i. e., networks on which physical objects such as cars, trains, or airplanes travel. More specifically:

- In “Engineering Route Planning Algorithms” Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner overview techniques to speed-up route planning algorithms and point out challenging variants of the problem.
- In From State of the Art Static Fleet Assignment to Flexible Stochastic Planning of the Future Sven Grothklags, Ulf Lorenz, and Burkhard Monien develop flexible strategies for aircraft assignment that can handle uncertainty in the input data at planning time.

- In "Traffic Networks and Flows Over Time" Ekkehard Köhler, Rolf H. Möhring, and Martin Skutella deal with optimization problems for traffic management and route guidance in street networks from the viewpoint of network flow theory.

The third part covers various algorithmic problems occurring in communication networks, i. e., networks such as the Internet on which information is sent. More specifically:

- In "Interactive Communication, Diagnosis and Error Control in Networks" Rudolf Ahlswede and Harout Aydinian survey results on connectors in communication networks, fault diagnosis in large multiprocessor networks, unconventional error-correcting codes, and parallel error-control codes.
- In "Resource Management in Large Networks" Susanne Albers reviews online strategies for buffer management in network routers and switches and online and offline strategies for Web caching where limited reordering of requests is allowed.
- In "Multicast Routing and Design of Sparse Connectors" Andreas Baltz and Anand Srivastav summarize studies of the minimum multicast congestion problem and describe efficient architectures for communication networks.
- In "Management of Variable Data Streams in Networks" Anja Feldmann, Simon Fischer, Nils Kammenhuber, and Berthold Vöcking show how and under which conditions interaction of selfish agents in communication networks leads to a Nash equilibrium, develop an algorithm to compute approximate equilibria, and design an online traffic engineering protocol.
- In "Models of Non-atomic Congestion Games – From Unicast to Multicast Routing" Lasse Kliemann and Anand Srivastav overview results for non-atomic congestion games and introduce and analyze *consumption-relevance congestion games* that comprise all other models.
- In "New Data Structures for IP Lookup and Conflict Detection" Christine Maindorfer, Tobias Lauer, and Thomas Ottmann present a survey of data structures for the representation of dynamic range router tables and outline improvements to online conflict detection.

The fourth part contains chapters on the analysis, visualization, and simulation of, among others, social networks or sensor networks. More specifically:

- In "Group-Level Analysis and Visualization of Social Networks" Michael Baur, Ulrik Brandes, Jürgen Lerner, and Dorothea Wagner present methods to define and compute structural network positions and techniques to visualize a network together with a given assignment of actors into groups.
- In "Modeling and Designing Real-World Networks" Michael Kaufmann and Katharina Zweig discuss models that explain how recurrent structures in real-world networks can emerge and treat the design of local network generating rules that lead to a globally satisfying network structure.

– In ‘Algorithms and Simulation Methods for Topology-Aware Sensor Networks” Alexander Kröller, Dennis Pfisterer, Sándor P. Fekete, and Stefan Fischer describe fundamental algorithmic issues for wireless sensor networks and present the network simulator *Shawn*.

We would like to thank all authors who submitted their work, as well as the referees for their invaluable contribution. The group of initiators is grateful to Kurt Mehlhorn for his helpful advice during preparation of the grant proposal. Finally, we would like to thank the DFG for accepting and sponsoring the priority program no. 1126 *Algorithmics of Large and Complex Networks*. We hope that this volume will serve as a starting point for further research in this important field.

March 2009

Jürgen Lerner
Dorothea Wagner
Katharina A. Zweig

Referees

Rudolf Ahlswede, Susanne Albers, Harout Aydinian, Ulrik Brandes, Sándor Fekete, Simon Fischer, Daniel Fleischer, Marco Gaertler, Horst Hamacher, Martin Hoefer, Michael Kaufmann, Lasse Kliemann, Jürgen Lerner, Martin Mader Rolf Möhring, Burkhard Monien, Matthias Müller-Hannemann, Rolf Niedermeier, Thomas Ottmann, Peter Sanders, Martin Skutella, Anand Srivastav, Siavash Vahdati-Daneshmand, Berthold Vöcking, Dorothea Wagner, Katharina Zweig

Table of Contents

Network Algorithms

Traffic Networks

Communication Networks

Network Analysis and Simulation

Design and Engineering of External Memory Traversal Algorithms for General Graphs

Deepak Ajwani[1] and Ulrich Meyer[2,*]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[2] Goethe-Universität, Frankfurt am Main, Germany

Abstract. Large graphs arise naturally in many real world applications. The actual performance of simple RAM model algorithms for traversing these graphs (stored in external memory) deviates significantly from their linear or near-linear predicted performance because of the large number of I/Os they incur. In order to alleviate the I/O bottleneck, many external memory graph traversal algorithms have been designed with provable worst-case guarantees. We describe the various techniques used in the design and engineering of such algorithms and survey the state-of-the-art in I/O-efficient graph traversal algorithms.

1 Introduction

Real world optimization problems often boil down to traversing graphs in a structured way. Graph traversal algorithms have therefore received considerable attention in the computer science literature. Simple linear time algorithms have been developed for breadth first search, depth first search, computing connected and strongly connected components on directed graphs, and topological ordering of directed acyclic graphs [25]. Also, there exist near-linear time algorithms for minimum spanning trees [21, 44, 55] on undirected graphs and single-source shortest paths [31,36] (SSSP) on directed graphs with non-negative weights. For all pair shortest paths (APSP), the naïve algorithm of computing SSSP from all nodes takes $O(m \cdot n + n^2 \log n)$, where n is the number of nodes and m is the number of edges. It has been improved to $O(m \cdot n + n^2 \log \log n)$ [54] for sparse graphs and $O(n^3 / \log n)$ [20] for dense graphs.

In many such applications, the underlying graph is too big to fit in the internal memory of the computing device. Consider the following examples:

- The world wide web can be looked upon as a massive graph where each webpage is a node and the hyperlink from one page to another is a directed edge between the nodes corresponding to those pages. As of August 2008, it is estimated that the indexed web contains at least 27 billion webpages [26]. Typical problems in the analysis (e.g., [14], [47]) of WWW graphs include computing the diameter of the graph, computing the diameter of the core

[*] Partially supported by the DFG grant ME 3250/1-1, DFG grant ME2088/1-3, and by MADALGO – Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 1–33, 2009.

of the graph, computing connected and strongly connected components and other structural properties such as computing the power in the power law modeling of WWW graphs. There has also been a lot of work on understanding the evolution of such graphs.

- The citations graph of all the scientific papers in the world from specific domains, where nodes are the publications and a directed edge from one paper to the other reflects a citation. The main problem here is to understand the nature of scientific collaboration and identify communities.
- Telephone call graphs: Telephone call graphs have the telephone numbers operated by a company as nodes and there is a directed edge between two nodes if and only if there has been a call from one number to another in a certain time-frame. The call graphs managed by telecom companies like AT&T can be massive. Typical problems on telephone call graphs are fraud detection and searching for local communities (e.g., by detecting maximum cliques [2]).
- GIS terrain data: Remote sensing has made massive amounts of high resolution terrain data readily available. Terrain analysis is central to a range of important geographic information systems (GIS) applications concerned with the effects of topography. Typical problems in this domain involve flow routing and flow accumulation [10].
- Route planning on small PDA devices [39, 59]: The PDA devices used to compute the fastest routes have very small main memory. Although the street maps of even continents are only a few hundred MBs in size, they are too large to fit into the small main memory of these devices.
- State space search in Artificial Intelligence [33]: In many applications of model checking, the state space that needs to be searched is too big to fit into the main memory. In these graphs, the different configuration states are the nodes and the edge between two nodes represent the possibility of a transition from one state to another in the course of the protocol/algorithm being checked. Typical problems in this domain are reachability analysis (to find out if a protocol can ever enter into a wrong state) [41], cycle detection (to search for liveness properties) [32], or just finding some path to eventually reach a goal state (action planning).

Unfortunately, as the storage requirements of the input graph reaches and exceeds the size of the internal memory available, the running time of the simple linear or near-linear time algorithms deviates significantly from their predicted asymptotic performance in the RAM model. On massive graphs (with a billion or more edges), these algorithms are non-viable as they require many *months or years* for the requisite graph traversal. The main cause for such a poor performance of these algorithms on massive graphs is the number of I/Os (transfer of data from/to the external memory) they incur. While it takes a few nano-seconds to access an element in the main memory, it usually takes a million times longer to fetch the same element from the disk.

Figure 1 displays the results of experiments with the commonly used BFS routine of the LEDA [50] graph package on random graphs [34, 35] $G(n, m)$

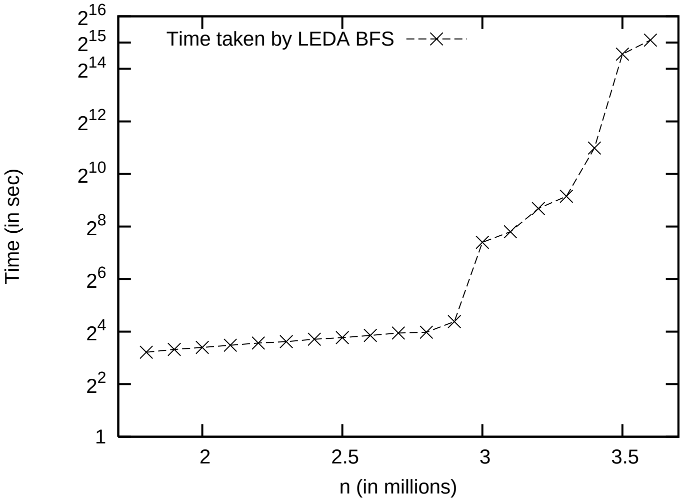


Fig. 1. Time (in seconds) required by BFS from the LEDA graph package on random graphs with $m = 4n$ edges

with $m = 4n$. These experiments were done on a machine with Intel Xeon 2.0 GHz processor, 1 GB RAM and 2 GB swap space on a Seagate Baracuda hard-disk [64]. On random graphs with 3.6 million nodes (and 14.4 million edges), it takes around 10 *hours* as compared to just 10 *seconds* for graphs with 1.8 million nodes (and 7.2 million edges).

In such situations, one often applies heuristics, pre-computations or exploits special properties of the underlying graphs. Such solutions are usually tailored for particular domains and are often application-specific. Ideally, we would like to have practical I/O-efficient algorithms for traversing general directed graphs.

1.1 External Memory Model

The huge difference in time between accessing an element from the main memory and fetching an element from the disk (I/O) is (nicely) captured by the external memory model (or the I/O model), which was introduced by Aggarwal and Vitter [3]. It assumes a single central processing unit and two levels of memory hierarchy (cf. Figure 2). The internal memory is fast, but has a limited size of M words. In addition, we have an external memory (consisting of D disks) which can only be accessed using I/Os that move $B \cdot D$ contiguous words between internal and external memory. For graph problems, the notation is slightly altered: we assume that the internal memory can have up to M data items of a constant size (e.g., nodes or edges), and in one I/O operation, $B \cdot D$ contiguous data items move between the two memories. At any particular time, the computation can only use the data already present in the internal memory. The measure of performance of an algorithm is the number of I/Os it performs. An algorithm incurring fewer number of I/Os is considered better.

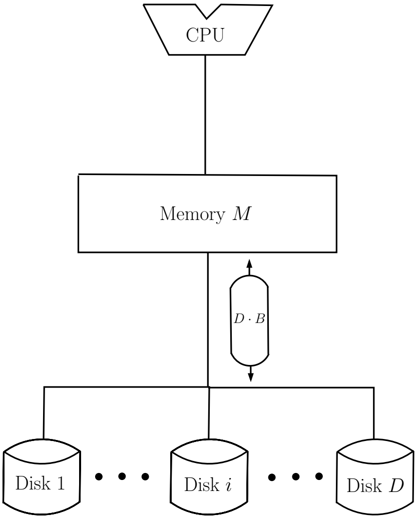


Fig. 2. The external memory model

In this paper, we will assume that the external memory consists of only one disk, i.e., $D = 1$. This assumption allows us to ignore the D term from the I/O complexities of the algorithms and simplify the presentation.

Typical values of B range around one million. As an example of what the different I/O complexities mean in terms of actual running time, consider a graph with 1 billion nodes and 10 billion edges stored on a Seagate Barracuda hard disk [64] (average latency: 4.2 msec). An algorithm incurring m I/Os will need around 1.3 *years*, one with n I/Os will require around 1.6 *months*, $n/\sqrt{B}$ I/O algorithm will need about 1.2 *hours* and the one with n/B I/Os requires only 4.2 *seconds*.

1.2 Cache-Oblivious Model

Modern computer architecture consists of a hierarchy of storage devices – registers, different levels (L1/L2/L3) of caches, main memory, and the hard disks – with very different access times and storage capacities. While external memory algorithms can be tuned for any two-level memory hierarchy by appropriately modifying the values of B and M, this tuning requires the knowledge of several hardware related parameters which are not always available from the manufacturer (particularly between the L2/L3 cache and the main memory) and are often difficult to extract automatically. Furthermore, parameter tuning makes code portability difficult. Ideally, we would like a model that captures the essence of memory hierarchy without knowing its specifics, i.e., values of B and M, and at the same time be efficient on all memory hierarchy levels simultaneously. Yet, it should be simple enough for a feasible and meaningful algorithm analysis. The cache oblivious model introduced by Frigo et al. [37] promises all of the above.

The cache-oblivious model also assumes a two level memory hierarchy with an internal memory of size M and block transfers of B elements in one I/O. The performance measure is the number of I/Os incurred by the algorithm. However, the algorithm does not have any knowledge of the values of M and B. Consequently, the guarantees on I/O-efficient algorithms in the cache-oblivious model do not only hold on any machine with multi-level memory hierarchy but also on all levels of the memory hierarchy at the same time. In principle, cache-oblivious algorithms are expected to perform well on different architectures without the need of any machine-specific optimization.

Note that any cache-oblivious algorithm is also an external memory algorithm with the same I/O complexity. On the other hand, cache-oblivious algorithms may have a worse I/O complexity than their external memory counterparts because of their generality. Also, the cache-oblivious algorithms (even those that have the same asymptotic bounds as their corresponding external memory algorithms) are usually slower in practice because of larger constant factors in their I/O complexity (see e.g., [5]).

1.3 Outline

In the last couple of decades, there has been a lot of work in the design of external memory (EM) and cache-oblivious algorithms for traversal problems on general graphs. Many algorithms with worst-case I/O guarantees have been designed. This has led to a better understanding of upper and lower bounds for the I/O complexity of many graph traversal problems.

Simultaneously, there has been work in engineering some of the more promising external memory and cache-oblivious algorithms into good implementations that can make graph traversal viable for massive graphs. Such engineering has been successful and has significantly extended the limits on the size of the underlying graphs that can be handled "in reasonable time".

This paper surveys the design and engineering of I/O-efficient graph traversal algorithms. In Section 2, we discuss various tools and techniques used in the design of these algorithms in external memory. Section 3 discusses the tools for engineering these algorithms. These include external memory libraries which play a crucial role in fast development and high performance of the implementations of these algorithms. Section 4 discusses the key problems in externalizing the simple linear or near-linear time internal memory algorithms. Sections 5, 6, 7, and 8 survey the problems of computing connected components/minimum spanning tree, BFS, SSSP, and APSP, respectively on *undirected* graphs. Section 9 covers the problem of A^* traversal in external memory. Section 10 describes the I/O-efficient *directed* graph traversal algorithms.

1.4 Notations and Graph Representation

In the rest of the paper, we will refer to the input graph as $G(V, E)$. A directed edge from u to v will be denoted by (u, v) and an undirected edge between u and v will be referred as $\{u, v\}$. For a directed edge (u, v), we call u the tail node and v the head node. We use the term adjacency list of a node u to refer to the set of all edges incident to u. Note that this term does not refer to the (pointer-based) list data structure.

In addition, we will use the following notations: $n = |V|$, $m = |E|$, d_v to refer to the degree of a node $v \in V$, and $d(s,t)$ to refer to the distance between the node s and t in G. For algorithms involving unweighted graphs such as list ranking, and BFS, the distance between two nodes is the number of edges on a minimum-length path between them. For weighted graphs (those arising in SSSP or APSP applications), we define the weight of a path P as the sum of the weights of the edges belonging to P. The distance between two nodes in this case is the minimum weight of any path between them.

For the problems considered in this paper, the input or the intermediate graphs are stored on the disk. The actual graph representation varies between different implementations, but in general it supports efficient retrieval of the adjacency lists of nodes (see e.g. [4]). Often, the nodes exist only virtually as consecutive numbers, and are not stored explicitly on the disk.

2 Tools and Techniques for Designing I/O-Efficient Graph Traversal Algorithms

Many different tools and techniques have been developed for graph algorithms in external memory in the last couple of decades. In this Section, we describe some of the commonly used building blocks for the design of I/O-efficient graph traversal algorithms.

2.1 Parallel Scanning

Scanning many different streams (of data from the disk) simultaneously is one of the most basic tools used in I/O-efficient algorithms. This can be used, for example, to copy some information from one stream to the other. Sometimes, different streams represent different sorted sets and parallel scanning can be used to compute various operations on these sets such as union, intersection, or difference.

Given $k = O(M/B)$ streams containing a total of $O(n)$ elements, we can scan them "in parallel" in $\text{scan}(n) := O(n/B + k)$ I/Os. This is done by simply keeping $O(1)$ blocks of each stream in the internal memory. When we need a block not present in the internal memory, we remove (or write back to the disk) the existing block from the corresponding stream and load the required block from the disk.

2.2 Sorting

Sorting is fundamental to many I/O-efficient graph traversal algorithms. In particular, sorting can be used to rearrange the nodes on the disk so that a graph traversal algorithm does not have to spend $\Omega(1)$ I/Os for loading the adjacency list of each node into the internal memory.

Sorting n elements in the external memory requires $\mathrm{sort}(n) := \Theta(\frac{n}{B} \cdot \log_{\frac{M}{B}} \frac{n}{B})$ I/Os [3]. There exist many different algorithms for I/O-efficient sorting. The most commonly used external memory sorting algorithm is based on (M/B)-way merge sort. It first scans through the input data, loading M elements at a time, sorting them internally and writing them back to disk. In the next round, we treat each of these chunks as a stream and merge $O(M/B)$ streams at a time using "parallel scanning" to produce sorted chunks of size $O(M^2/B)$. By repeating this process for $O(\log_{\frac{M}{B}} \frac{n}{B})$ rounds, we get all the elements sorted.

External memory libraries such as STXXL [28,29] and TPIE [12] provide fast implementations of external memory sorting routines. STXXL also has specialized functions for sorting elements with integer keys and sorting streams.

In the cache-oblivious setting, funnel-sort [37] and lazy funnel-sort [15], also based on a merging framework, lead to sorting algorithms with the same I/O complexity of $\Theta(\frac{n}{B} \log_{\frac{M}{B}} \frac{n}{B})$ I/Os. Brodal et al. [18] show that a careful implementation of this algorithm outperforms several widely used library implementations of quick-sort on uniformly distributed data. For the largest instances in the RAM, this implementation outperforms its nearest rival std::sort from the STL library included in GCC 3.2 by 10-40% on many different architectures like Pentium III, Athlon and Itanium 2.

2.3 PRAM Simulation

A Parallel Random Access Machine (PRAM) is a basic model of computation that consists of a number of sequential processors, each with its own memory, working synchronously and communicating between themselves through a common shared memory. Simulating a PRAM algorithm [23] on the external memory model is an important tool in the design of I/O-efficient graph algorithms. A PRAM algorithm that uses p processors and $O(p)$ (shared memory) space and runs in time $T(p)$ can be simulated in $O(T(p) \cdot \mathrm{sort}(p))$ I/Os.

Each step taken by a PRAM involves each processor independently reading a data element, computing on it and writing some output. In order to simulate it on the external memory model, the read requests of all the processors are sorted according to the location of the required data. Afterwards, one scan of the entire data of the shared memory is enough to fetch all the requisite data. This is then sorted back according to the processor ids. Thereafter, in one scan of the fetched data, we perform all the computations by all the processors and collect the output data (together with its location) that would have been produced by each processor. This is then sorted according to the memory location and written back to the disk. Thus, each step of the $O(p)$-processor PRAM algorithm requiring $O(p)$ space can be simulated by a constant number of sorts and scans, i.e., $O(\mathrm{sort}(p))$ I/Os.

PRAM simulation is particularly appealing as it translates a large number of PRAM-algorithms into I/O-efficient and sometimes I/O-optimal algorithms.

Even without directly using the simulation, I/O-efficient algorithms can be obtained by appropriately translating PRAM algorithms, as many of the ideas applied in parallel computing for reducing a problem into many independent sub-problems are also useful for designing external memory algorithms. For many problems, the bounds obtained by appropriately translating PRAM algorithms are much better than those obtained by direct simulation.

2.4 Algorithms on Trees

Efficient external memory algorithms are known for many different problems on undirected trees. These include rooting a tree, computing pre-order, post-order or in-order traversal, computing the depth of each node, least common ancestor queries, etc. Most of these algorithms (e.g., the tree traversal algorithms in [23]) are efficient translations of their PRAM counterparts.

2.5 Euler Tour

An Euler tour of an undirected tree $T = (V, E)$ is a traversal of T that traverses every edge exactly twice, once in each direction. Such a traversal produces a linear list of edges or vertices capturing the structure of the tree. In order to compute such a tour, we choose an order of the edges $\{v, w_1\}, \ldots, \{v, w_k\}$ incident to each node v of T. Then, we mark the successor of $\{w_i, v\}$ to be $\{v, w_{i+1}\}$ and the successor of $\{w_k, v\}$ to be $\{v, w_1\}$. We break the resultant circular list at some node r by choosing an edge $\{v, r\}$ with successor $\{r, w\}$, setting the successor of $\{v, r\}$ to be null and choosing $\{r, w\}$ to be the first edge of the traversal. An Euler tour of a tree can thus be computed in $O(\text{sort}(n))$ I/Os.

2.6 Priority Queues

A priority queue is an abstract data structure that stores an ordered set of keys and allows for efficient insertion, search of the minimum element (find_min) and deletion of the minimum element (delete_min). Sometimes operations such as deleting an arbitrary key and decreasing the value of the key are also supported. Priority queues are fundamental to many graph traversal algorithms, particularly for computing single-source shortest-paths.

One way of implementing efficient external memory priority queues is using buffer trees [7]. Buffer trees are useful for batched operations, i.e., when the answers to the queries are not required immediately but eventually.

A buffer tree has degree $\Theta(M/B)$. Each internal node is associated with a buffer containing a sequence of up to $\Theta(M)$ updates and queries to be performed in its subtree. Leaves contain $\Theta(B)$ keys. Updates and queries are simply performed by inserting the appropriate signal in the root node buffer. If the buffer is full, it is flushed to its children. This process may need to be repeated all the way down to the leaves. Since flushing the buffer requires $\Theta(M/B)$ I/Os (which is done after inserting $\Theta(M)$ signals) and the tree has $O(\log_{M/B} \frac{n}{B})$ levels, the amortized cost of the update and query operations is $O(\frac{1}{B} \cdot \log_{\frac{M}{B}} \frac{n}{B})$ I/Os. It can be shown that the re-balancing operations for maintaining the tree can also be done within the same bounds.

In order to use buffer trees as a priority queue, the entire buffer of the root node together with the $O(M/B)$ leftmost leaves (all the leaves of the leftmost internal node) is kept in internal memory. We maintain the invariant that all buffers on the path from the root to the leftmost leaf are empty. Thus, the element with the smallest priority always remains in internal memory. The invariant is maintained by flushing out all buffers in the leftmost path whenever the root buffer is flushed, at a total cost of $O(\frac{M}{B} \cdot \log_{\frac{M}{B}} \frac{n}{B})$ I/Os. The amortized cost of updates and queries still remains $O(\frac{1}{B} \cdot \log_{\frac{M}{B}} (\frac{n}{B}))$ I/Os.

Note that the buffer tree based priority queue can not efficiently perform a decrease_key of an element, if we do not know its old key. For efficient but lazy decrease_key operations, we can use tournament trees [46]. On an I/O-efficient tournament tree with n elements, any sequence of z operations each of them being either a delete, delete_min or an update, requires at most $O(\frac{z}{B} \cdot \log_2 \frac{n}{B})$ I/Os. The update operation referred here is a combined insert and decrease_key operation.

Cache-oblivious priority queues with amortized $O(\frac{1}{B} \cdot \log_{\frac{M}{B}} \frac{n}{B})$ I/O insertion, deletion and delete_min operations have also been developed [8, 17]. The cache-oblivious bucket-heap based priority queue [16] provides amortized $O(\frac{1}{B} \cdot \log_2 \frac{n}{B})$ update, delete and delete_min operations, where the update operation is a combined insert and decrease_key operation similar to the one provided by tournament trees.

2.7 Time Forward Processing

Time forward processing [7,23] is an elegant technique for solving problems that can be expressed as a traversal of a directed acyclic graph (DAG) from its sources to its sinks. Let G be a DAG and $\phi(v)$ be a label associated with the node v. The goal is to compute another labelling $\psi(v)$ for all nodes $v \in G$, given that $\psi(v)$ can be computed from labels $\phi(v)$ and $\psi(u_1), \ldots, \psi(u_k)$, where $u_1, \ldots, u_k$ are the in-neighbors of v.

Time forward processing on an n-node DAG can be solved in external memory in $O(\text{sort}(n))$ I/Os if the following conditions are met:

1. The nodes of G are stored in topologically sorted order.
2. $\psi(v)$ can be computed from $\phi(v)$ and $\psi(u_1), \ldots, \psi(u_k)$ in $O(\text{sort}(k))$ I/Os.

This bound is achieved by processing the nodes in the topologically sorted order and letting each node pass its label ψ to its out-neighbors using a priority queue. Each node u inserts $\psi(u)$ in the priority queue for each out-neighbor v with the key being the topological number of v, $T(v)$. We ensure that before we process v, we extract all the nodes with priority $T(v)$ and therefore, get all the necessary information to compute $\psi(v)$.

2.8 List Ranking

A list L is a collection of elements $x_1, \ldots, x_n$ such that each element x_i, except the last element of the list, stores a pointer to its successor, no two elements have the same successor and every element can reach the last element by following successor pointers. Given a list L of elements kept in an arbitrary order on the disk and a pointer to the first element and weights w on all edges, the list ranking problem is that of computing for every element x_i, its distance from the first element.

The external memory list ranking algorithm [23] computes an independent set I of size $\Omega(n)$. All elements $x_i \in I$ are removed from L by marking $succ(x_i)$ as the successor of $pred(x_i)$, where $succ(x_i)$ and $pred(x_i)$ are the successor and predecessor of x_i in L. The weight of the new edge $\{pred(x_i), succ(x_i)\}$ is the sum of the weights of $\{pred(x_i), x_i\}$ and $\{x_i, succ(x_i)\}$. The problem on the compressed list is recursively solved. For each node $x_i \in I$, its distance from the head is equal to the sum of the distance of $pred(x_i)$ (computed for the compressed list) and the weight of the edge $\{pred(x_i), x_i\}$. All operations for compressing the list incur $O(\text{sort}(n))$ I/Os and thus the total cost of list ranking is $I(n) = I(\alpha \cdot n) + O(\text{sort}(n)) = O(\text{sort}(n))$ I/Os, for some constant $0 < \alpha < 1$.

Note that any maximal independent set of a list has size at least $n/3$. Thus in order to compute the independent set I of size $\Omega(n)$, we just need to compute a maximal independent set. A maximal independent set I of a graph $G(V, E)$ can be computed simply by a greedy algorithm in which the nodes are processed in an arbitrary order. When a node $v \in V$ is visited, we add it to the set I if none of its neighbors is already in I. This can be done in $O(\text{sort}(n + m))$ I/Os using time forward processing.

A list of length n can thus be ranked in $O(\text{sort}(n))$ I/Os.

List ranking algorithm in practice. Another list ranking algorithm due to Sibeyn [60] has low constant factors (for realistic input size) in its I/O complexity and is therefore, more practical. The algorithm splits the input list into sublists of size $O(M)$ and goes through the data in a wave-like manner. For all elements of the current sublist, it follows the links leading to the elements of the same sublist and updates the information on their final element and the number of links to it. For all elements with links running outside the current sublist, the required information is requested from the sublists containing the elements to which they are linked. The requests and the answers to the sublists are stored in one common stack and processed only when the wave through the data hits the corresponding sublist. *Bucketing* and *lazy processing* of the requests and the answers to the sublists make the implementation [60] faster than the algorithms based on independent set removal by a factor of about four. The implementation [5] of this algorithm in the STXXL framework is quite fast in practice and has been used as a building block in the engineering of many graph traversal algorithms.

2.9 Graph Contraction

The key idea in graph contraction is to reduce the size of the input graph G while preserving the properties of interest. Such a procedure is often applied recursively till either the number of edges or the number of nodes are reduced by a factor of $O(B)$ or the number of nodes is reduced to $O(M)$. In the first case, the algorithm can afford to spend $O(1)$ I/Os per remaining node to solve the problem. In the latter case, an efficient semi-external algorithm is used to solve the problem.

Graph contraction is particularly useful for problems like connected components and minimum spanning forests, where the connectivity information is preserved (see e.g. [9]) during the edge contraction steps.

2.10 Graph Clustering

Clustering a graph refers to decomposing the graphs into disjoint clusters of nodes. Each cluster contains the adjacency lists of a few nodes. These nodes should be close in the original graph. Hence, if a node of the cluster is visited during BFS, SSSP or APSP, the other nodes of the cluster will also be visited "shortly". This fact can be exploited (see e.g., [49,51]) to design better algorithms for these problems.

2.11 Ear Decomposition

An ear decomposition $\varepsilon = (P_0, P_1, P_2, \ldots, P_k)$ of a graph $G = (V, E)$ is a partition of E into an ordered collection of edge-disjoint simple paths P_i with endpoints s_i and t_i. Ear P_0 is an edge. For $1 \leq i \leq k$, ear P_i shares its two endpoints s_i and t_i, but none of its internal nodes, with the union $P_0 \cup \ldots P_{i-1}$ of all previous ears. A graph has an ear decomposition if and only if it is two-edge connected, i.e., removing any edge still leaves a connected subgraph.

An ear decomposition of a graph can be computed in $O(\text{sort}(n))$ I/Os in external memory [48].

3 Tools for Engineering External Memory Graph Traversal Algorithms

In the last decade, many techniques have evolved for engineering external memory graph traversal algorithms. Libraries specifically containing fundamental algorithms and data structures for external memory have been developed. Techniques such as pipelining can save some constant factors in the I/O complexity of the external memory implementations, which can be significant for making the implementation viable. In this section, we describe some of these tools and techniques.

3.1 External Memory Libraries

External memory libraries play a crucial role in engineering algorithms running on large data-sets. These libraries not only reduce the development time for external memory algorithms, but also speed up the implementations themselves. The former is done by abstracting away the details of how an I/O is performed and providing ready-to-use building blocks including algorithms such as sorting and data structures such as priority queues. The latter is done by offering frameworks such as pipelining (c.f. Section 3.2) that can reduce the constant factors in the I/O complexity of an implementation. Furthermore, the algorithms and data structures provided are optimized and perform less internal memory work.

STXXL. STXXL [28, 29] is an implementation of the C++ standard template library STL [62] for external memory computations. Since the data-structures and algorithms in STXXL have a well-known generic interface similar to that of STL, it is easy to use and the existing applications based on STL can be easily made to work with STXXL. STXXL supports parallel disks, overlapping between disk I/O and computation and the *pipelining* technique that can save a significant fraction of the I/Os. It provides I/O-efficient implementations of various containers (stack, queue, deque, vector, priority queue, B^+-tree, etc.) and algorithms (scanning, sorting using parallel disks, etc.). It is being used both in academic and industrial environments for a range of problems including text processing, graph algorithms, computational geometry, Gaussian elimination, visualization, and analysis of microscopic images, differential cryptographic analysis, etc. Among graph algorithms in particular, it has been used for implementing external memory minimum spanning trees, connected components, breadth-first search, and social network analysis metrics.

TPIE. TPIE [12] or “Transparent Parallel I/O Environment” is another C++ template library supporting out-of-core computations. The goal of the TPIE project has been to provide a portable, extensible, flexible, and easy to use programming environment for efficiently implementing I/O-efficient algorithms and data structures. Apart from supporting algorithms with a sequential I/O pattern (i.e., algorithms using primitives such as scanning, sorting, merging, permuting and distributing) and basic data structures such as B^+-tree, it supports many more external memory data structures such as (a, b)-tree, persistent B-tree, Bkd-tree, K-D-B-tree, R-tree, EPS-tree, CRB-tree etc. It is used for many geometric and GIS implementations.

3.2 Pipelining

Conceptually, pipelining is a partitioning of the algorithm into practically independent parts that conform to a common interface, so that the data can be streamed from one part to the other without any intermediate external memory storage. This may reduce the constant factors in the I/O complexity of the algorithm. It leads to better structured implementations, as different parts of the pipeline only share a narrow common interface. On the other hand, it may also increase the computational costs as in a stream, searching an element can't be done by exponential or binary search, but by going through potentially all the elements in the stream. This means that the correct extent of pipelining needs to be carefully determined.

Usually, a pipelined code requires more debugging efforts and hence, significantly larger development time. For more details on the usage of pipelining as a tool to save I/Os, refer to [27].

3.3 Heuristics

The external memory implementations also use various heuristics that preserve the theoretical guarantees of the basic algorithms. Often, these heuristics try to exploit the available resources better – be it main memory or the computation. As an example, Section 6.3 shows some heuristics for the external memory BFS implementations that help to use the main memory more efficiently.

4 Key Problems in EM Graph Traversal

Before describing the design and engineering of various external memory graph traversal algorithms, let us look into the following problems associated with running internal memory graph algorithms for computation on externally stored graphs:

1. Remembering visited nodes needs $\Theta(m)$ I/Os in the worst case.
2. Unstructured accesses to adjacency lists (or random I/Os to load adjacency lists into internal memory) may result in $\Theta(n)$ I/Os.
3. Lack of an efficient addressable external memory priority queue. Most of the external memory priority queue data structures rely on the concept of lazy batched processing. This means that decrease_keys and deletes need not take effect immediately. (However, the correctness of the delete_min operation is not compromised.) Moreover, they do not support decrease_key operation in $O(\frac{1}{B})$ I/Os. Furthermore, the new key can not depend on the old key in any way.

The first problem can be solved by using data structures such as a buffered repository tree [19] and priority queues to remove the edges that lead to already settled nodes. In the case of single-source shortest paths, some algorithms even allow the already visited or settled nodes to enter the priority queue again, but then they take care to remove them before they can lead to any inaccuracy in the results. For BFS on undirected graphs, this problem can be solved by exploiting the fact that the neighbors of a node in BFS level t are all in level $t-1$, t or $t+1$.

Solving the second problem is usually more tricky. For BFS on undirected graphs, this is partially solved by re-arranging the graph layout on the disk. It is not clear whether and how it can be solved for other graph traversal problems.

5 Minimum Spanning Forest

Given an undirected connected graph G, a spanning tree of G is a tree-shaped subgraph that connects all the nodes. A minimum spanning tree is a spanning tree with minimum total sum of edge weights. For a general undirected graph (not necessarily connected), we define a minimum spanning forest (MSF) to be the union of the minimum spanning trees for its connected components. Computing a minimum spanning forest of a graph G is a well-studied problem in the RAM model.

The first algorithm for this problem is due to Boruvka [13]. This algorithm runs in phases; in each phase we find the lightest edge incident to each node. These edges are output as a part of the MSF. Contracting these edges leads to a new graph with at most half of the nodes. Since the remaining MSF edges are also in the MSF of the contracted graph, we recursively output the MSF edges of the contracted graph.

The most popular algorithms for MSF in the RAM model are Kruskal's and Prim's algorithms. Kruskal's algorithm [45] looks at the edges in increasing order of their weight and maintains the minimum spanning forest of the edges seen so far. A new edge is output as a part of the MSF if its two endpoints belong to different components in the current MSF. The necessary operations can be performed efficiently using a disjoint set (union-find) data structure [38]. The resultant complexity for this algorithm is $O(n \cdot \alpha(n))$ [63], where $\alpha(\cdot)$ is the inverse Ackermann function.

Unlike Kruskal's algorithm which maintains potentially many different MSTs at the same time, Prim's algorithm [43, 56] works by "growing" one MST at a time. Starting with an arbitrary node, it searches for the lightest edge incident to the current tree and outputs it as a part of the MST. The other endpoint of the edge is then added to the current tree. The candidate edges are maintained using Fibonacci heaps [36], leading to an asymptotic complexity of $O(m + n \log n)$. If there is no edge between a node inside and a node outside the current MST, we "grow" a new MST from an arbitrarily chosen node outside the MSF "grown" so far.

5.1 Semi-external Kruskal's Algorithm

In the semi-external version of Kruskal's algorithm, an external memory sorting algorithm is used to sort the edges according to their edge weights. The minimum spanning forest and the union-find data structure are kept in the internal memory (as both require $O(n)$ space). The I/O complexity of this algorithm is $O(\text{sort}(m))$ under this setting.

5.2 External Memory Prim's Algorithm

In order to externalize Prim's algorithm, we use an external memory priority queue (cf. Section 2.6) for maintaining the set of candidate edges to grow the current minimum spanning tree. This results in an I/O complexity of $O(n + \text{sort}(m))$. The $O(n)$ term comes from the unstructured accesses to the adjacency lists, as we spend $O(1 + d_v/B)$ (d_v being the degree of v) I/Os to get hold of edges incident to the node v that need to be inserted into the priority queue.

5.3 External Memory Boruvka Steps

In most external memory algorithms, a Boruvka step like contraction method is used to reduce the number of nodes to either $O(M)$ or $O(m/B)$. In the first case, semi-external Kruskal's algorithm or other semi-external base cases are used. In the latter case, any external algorithm like Prim's algorithm or BFS (described in Section 6) can be used as we can afford one I/O per node in the contracted graph.

We initialize the adjacency lists of all nodes by sorting the edges first according to their tail node and that being equal, by their weight. In each EM Boruvka phase, we find the minimum weight edge for each node and output it as a part of MSF. This can easily be done by scanning the sorted adjacency lists. This partitions the nodes into pseudo-trees (a tree with one additional edge). The minimum weight edge in each pseudo-tree is repeated twice, as it is the minimum weight edge incident to both its end-points. Such edges can be identified in $O(\text{sort}(m))$ I/Os. By removing the repeated edges, we obtain a forest. We select a leader for each tree in the forest and let each node $u \in V$ know the leader $L(u)$ of the tree containing it. This can be done by variants of external memory list ranking algorithm (cf. Section 2.8) or by using time forward processing (cf. Section 2.7) in $O(\text{sort}(n))$ I/Os. We then replace each edge (u, v) in E by an edge $(L(u), L(v))$. At the end of the phase, we remove all isolated nodes, parallel edges and self loops. Again, this requires a constant number of sorts and scans of the edges.

The Boruvka steps as described here reduce the number of nodes by at least a factor of two in one phase and each phase costs $O(\text{sort}(m))$ I/Os. Thus, it takes $\log \frac{n \cdot B}{m}$ phases to reduce the number of nodes to $O(m/B)$, after which the externalized version of Prim's algorithm or BFS algorithm can be used. This gives a total I/O complexity of $O(\text{sort}(m) \cdot \log \frac{n \cdot B}{m})$. Alternatively, we can have $O(\log \frac{n}{M})$ phases of Boruvka's algorithm to reduce the number of nodes to $O(M)$ in order to apply semi-external version of Kruskal's algorithm afterwards. This will result in a total I/O complexity of $O(\text{sort}(m) \cdot \log \frac{n}{M})$.

5.4 An $O(\text{sort}(m) \cdot \log\log(\frac{n \cdot B}{m}))$ I/O Algorithm

Arge et. al. [9] improved the asymptotic complexity of the above algorithm by dividing the $O(\log \frac{n \cdot B}{m})$ phases of Boruvka steps into $O(\log\log \frac{n \cdot B}{m})$ super-phases requiring $O(\text{sort}(m))$ I/Os each. The idea is that rather than selecting only one edge per node, we select $\sqrt{S_i}$ lightest edges for contraction in each super-phase, where $S_i := 2^{(3/2)^i} (= S_{i-1}^{3/2})$. If a node does not have that many adjacent edges, all its incident edges are selected and the node becomes inactive. The selected edges form a graph G_i. We apply $\log \sqrt{S_i}$ phases of Boruvka steps on G_i to compute a leader $L(u)$ for each node $u \in V$. At the end of the super-phase, we replace each edge (u, v) in E by an edge $(L(u), L(v))$ and remove isolated nodes, parallel edges and self loops.

The number of active nodes after phase i is at most $n/(S_i \cdot S_{i-1} \cdots S_0) = n/(S_i \cdot S_i^{2/3} \cdots S_0) \le n/S_i^{5/3} \le n/S_{i+1}$ and thus $O(\log\log \frac{n \cdot B}{m})$ super-phases suffice to reduce the number of nodes to $O(n \cdot B/m)$.

Note that in super-phase i, there are $\log \sqrt{S_i}$ phases of Boruvka steps on G_i. Since G_i has at most n/S_i nodes at the beginning of phase i and $n\sqrt{S_i}/S_i$

edges (as each of the n/S_i nodes selects $\sqrt{S_i}$ edges around it), total cost of all these Boruvka phases is $O(\text{sort}(n/\sqrt{S_i}) \cdot \log \sqrt{S_i}) = O(\text{sort}(n))$ I/Os. The cost of replacing the edges by the contracted edges and other post-processing is $O(\text{sort}(m))$ I/Os.

Since each super-phase takes $O(\text{sort}(m))$ I/Os, the total I/O complexity of the algorithm is $O(\text{sort}(m) \cdot \log\log(\frac{n \cdot B}{m}))$.

Arge et al. [8] propose a cache-oblivious minimum spanning tree algorithm that uses a cache-oblivious priority queue to achieve the I/O complexity of $O(\text{sort}(m) \cdot \log\log n)$.

Connected components. Minimum spanning forest also contains the information regarding the connected components of the graph. For directly computing connected components, one can use the above algorithm by modifying the comparator function for edge weights (since the weights on the edges can be ignored for connected components computation) – an edge is smaller than the other edge if either the head node has a smaller index or the two head nodes are equal, but the tail node has a smaller index.

5.5 Randomized CC and MSF

Abello et. al. [1] proposed a randomized algorithm for computing connected components and minimum spanning tree of an undirected graph in external memory in $O(\text{sort}(m))$ expected I/Os. Their algorithm uses Boruvka steps together with edge sampling and batched least common ancestor (LCA) queries in a tree.

5.6 Experimental Analysis

Dementiev et al. [30] carefully engineered an external memory MSF algorithm. Their implementation is based on a sweeping paradigm to reduce the number of nodes to $O(M)$ and then running the semi-external Kruskal's algorithm. The node contraction phase consists of repeatedly choosing a node at random and contracting the lightest edge incident to it. In external memory, selecting random nodes can be done by using I/O-efficient random permutation (e.g., [58]) and looking at the nodes in that order. In contracting the edges, one needs to "inform" all the other neighbors of the non-leader node about the leader node. This can be done by time-forward processing (cf. Section 2.7) using external memory priority queues. This MSF implementation uses STXXL for sorting, priority queue and other basic data structures.

Dementiev et al. [30] showed that with their tuned implementation, massive minimum spanning tree problems filling several hard disks can be solved "overnight" on a low cost PC-server with 1 GB RAM. They experimented with many different graph classes – random graphs, random geometric graphs and grid graphs. In general, they observed that their implementation of the semi-external version of Kruskal's algorithm only loses a factor of two in the execution time per edge as compared to the internal memory algorithm. Their external memory implementation merely loses an additional factor of two.

6 Breadth-First Search

Given a large undirected graph $G(V, E)$ and a source node s, the goal of Breadth-First Search (BFS) is to decompose the set of nodes V into disjoint subsets called BFS levels, such that the level i comprises of all nodes that can be reached from s via i edges, but no less. BFS is well-understood in the RAM model. There exists a simple linear time algorithm [25] (hereafter referred as IM_BFS) for the BFS traversal in a graph. IM_BFS visits the nodes in a one-by-one fashion; appropriate candidate nodes for the next node to be visited are kept in a FIFO queue Q. After a node v is extracted from Q, the adjacency list of v is examined in order to append unvisited nodes to Q, and the node v is marked as visited.

However, this algorithm performs quite badly when the input graph does not fit into the main memory. The algorithm can incur $\Omega(m)$ I/Os for remembering visited nodes and $\Omega(n)$ I/Os for accessing adjacency lists.

6.1 Munagala and Ranade's Algorithm

The algorithm by Munagala and Ranade [53] (MR_BFS) solves the first problem by exploiting the fact that in an undirected graph, the edges from a node in BFS level t lead to nodes in BFS levels $t-1$, t or $t+1$ only. Thus, in order to compute the nodes in BFS level $t+1$, one just needs to collect all neighbors of nodes in level t, remove duplicates and remove the nodes visited in levels $t-1$ and t (as depicted in Figure 3). Since all these steps can be done in $O(\text{sort}(n))$ I/Os, the total number of I/Os required by this algorithm is $O(n + \text{sort}(m))$. The $O(n)$ term comes from unstructured accesses to adjacency lists.

6.2 Mehlhorn and Meyer's Algorithm

In order to solve the problem of unstructured accesses to adjacency lists, Mehlhorn and Meyer [49] (MM_BFS) propose a pre-processing step in which the input graph is rearranged on the disk. The preprocessing phase involves clustering the input graph into small subgraphs of low diameter. The edges of a cluster are stored contiguous on the disk. This is useful as once a node from the cluster is visited, other nodes in the cluster will also be visited soon (owing to low diameter of

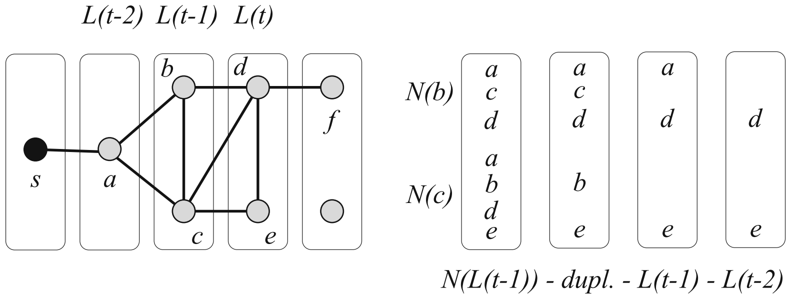


Fig. 3. A phase in the BFS algorithm of Munagala and Ranade

the cluster). By spending only one random access (and possibly, some further sequential accesses depending on the cluster size) for loading the whole cluster and then keeping the cluster data in some efficiently accessible data structure (hot pool) until it is all used up, on sparse graphs, the total number of I/Os can be reduced by a factor of up to $\sqrt{B}$. The neighboring nodes of a BFS level can be computed simply by scanning the hot pool and not the whole graph. Though some edges may be scanned multiple times in the hot pool, unstructured I/Os for fetching adjacency lists are considerably reduced, thereby limiting the total number of I/Os.

The input graph is decomposed into $O(n/\mu)$ clusters of diameter $O(\mu)$ for some parameter μ to be fixed later. This can be done in two ways – one based on "parallel cluster growing" and the other based on Euler tours.

Parallel cluster growing variant. The "parallel cluster growing" variant (MM_BFS_R) of MM_BFS works by choosing master nodes independently and uniformly at random with probability $O(1/\mu)$. The source node s is also chosen to be a master node. Thereafter, we run a local BFS from all master nodes "in parallel". In each round, each master node tries to capture all unvisited neighbors of its current sub-graph. The ties can be resolved arbitrarily.

Capturing new nodes on the fringes of all clusters can be done by sorting the neighbors of the nodes captured in the previous round and then scanning the adjacency lists of the input graph. Each round i thus takes $O(\text{sort}(n_i)+\text{scan}(m))$ I/Os, where n_i is the number of nodes captured in round $i-1$. The expected number of master nodes is $O(1+n/\mu)$ and the number of rounds (number of edges in a shortest path between any node and its cluster center) is $O(\log n \cdot \mu)$ with high probability (w.h.p.). Thus the total complexity for this clustering is $O(\text{sort}(n+m)+\text{scan}(n+m)\cdot\mu\cdot\log n)$ w.h.p. and it produces $O(n/\mu)$ clusters of diameter $O(\log n \cdot \mu)$ w.h.p.

Euler tour based clustering. In this variant (MM_BFS_D), we first use a connected components algorithm to identify the component of the graph containing the source node s. The nodes outside this component are output with BFS level ∞ and can be safely ignored, as they do not affect the BFS level of any other node. Then, we compute a spanning tree of nodes in this connected component. Considering the undirected edges of this tree as bi-directional edges (cf. Figure 4), we compute an Euler tour on these edges. Note that this has size at most $2n-2$. We then employ the list ranking algorithm to store the nodes on the disk in the order of their appearance in the Euler tour (some nodes may appear multiple times). The nodes arranged in this way are then chopped into $\frac{2n-2}{\mu}$ clusters of size μ. After removing the duplicates from this sequence, we get the requisite clustering of nodes.

Since CC/MST can be computed in $O((1+\log\log\frac{n\cdot B}{m})\cdot\text{sort}(n+m))$ I/Os and the Euler tour and list ranking of $O(n)$ elements can both be done in $O(\text{sort}(n))$ I/Os, the total complexity of this preprocessing is $O((1+\log\log\frac{n\cdot B}{m})\cdot\text{sort}(n+m))$ I/Os.

BFS phase. The actual BFS computation is similar to the algorithm by Munagala and Ranade, but with one crucial difference: the adjacency lists of

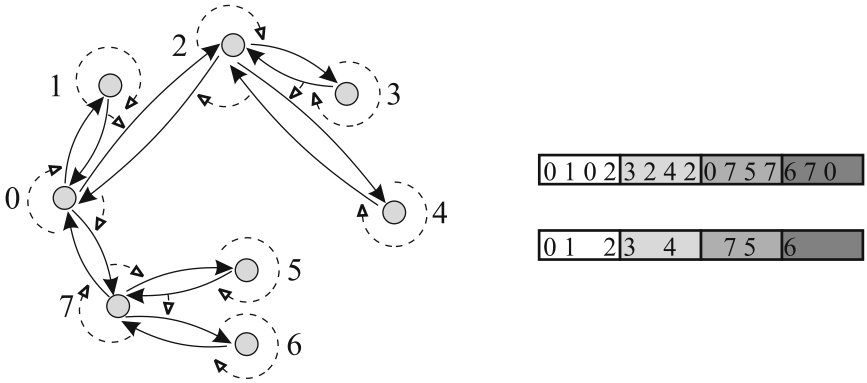


Fig. 4. The bi-directional tree (solid lines) and the Euler tour around it (dashed lines) on the left. The order of the nodes and their partitioning before and after the duplicates removal on the right.

nodes in the current level are no longer accessed directly from the input graph using random I/Os. Instead, the nodes in BFS level $t-1$ are scanned in parallel with the nodes in the hot pool H to determine if the adjacent edges of the former set are in H or not. If not, the entire clusters containing any of the nodes in level $t-1$ are merged into H. A next round of scanning the nodes in BFS level $t-1$ in parallel with the hot pool H fetches all the requisite adjacency lists.

Since each cluster is merged exactly once, it requires $O(n/\mu + \text{scan}(m))$ I/Os to load these clusters into H. For the Euler-tour-based approach, each adjacency list in H is scanned for at most $O(\mu)$ rounds as the distance between any two nodes in the cluster is $O(\mu)$. Thus the total number of I/Os required for the BFS phase by the Euler-tour-based variant of MM_BFS is $O(n/\mu + \mu \cdot \text{scan}(n+m) + \text{sort}(n+m))$. By choosing $\mu = \max\left\{1, \sqrt{\frac{n}{\text{scan}(n+m)}}\right\}$, we get a total I/O complexity (including the pre-processing) of $O(\sqrt{n \cdot \text{scan}(n+m)} + \text{sort}(n+m) + ST(n,m))$ I/Os for MM_BFS_D, where $ST(n,m)$ is the number of I/Os required to compute a spanning tree (of the connected component containing the source node) of a graph with n nodes and m edges. Using the randomized algorithm for MST/CC with $O(\text{sort}(n+m))$ expected I/O complexity, MM_BFS_D requires expected $O(\sqrt{n \cdot \text{scan}(n+m)} + \text{sort}(n+m))$ I/Os.

For the "parallel cluster growing" variant, an adjacency list stays in H for $O(\mu \cdot \log n)$ levels w.h.p. Since there are at most $1 + \frac{n}{\mu}$ clusters and each cluster is loaded at most once, loading them into H requires $O(\frac{n}{\mu} + \text{scan}(m))$ I/Os. The total complexity for MM_BFS_R is thus $O(n/\mu + \mu \cdot \log n \cdot \text{scan}(n+m) + \text{sort}(n+m))$ I/Os w.h.p. Choosing $\mu = \max\left\{1, \sqrt{\frac{n}{\text{scan}(n+m) \cdot \log n}}\right\}$, we get an I/O complexity of $O(\sqrt{n \cdot \text{scan}(n+m) \cdot \log n} + \text{sort}(n+m))$ I/Os w.h.p. for MM_BFS_R.

Brodal et al. [16] gave a cache-oblivious undirected BFS algorithm that has a complexity of $O(\mathrm{sort}(n+m) + \mathrm{scan}(m) \cdot \log n + \sqrt{n \cdot \mathrm{scan}(n+m)} + ST(n,m))$ I/Os, where $ST(n,m)$ is the complexity of computing a spanning tree of a graph with n nodes and m edges in a cache-oblivious way. The current best cache-oblivious algorithm for computing a spanning tree requires $O(\mathrm{sort}(m) \cdot \log\log n)$ I/Os deterministically and $O(\mathrm{sort}(m))$ I/Os randomized. While the I/O complexity of this cache-oblivious undirected BFS algorithm is quite close to the complexity of the best I/O-model algorithm for undirected BFS, it is likely to be slower in practice [5].

6.3 Engineering EM BFS

Ajwani et al. [4,5] implemented MR_BFS and MM_BFS using the external memory library STXXL. Their pipelined implementation exploits disk parallelism and uses fast stream sorters (based on M/B-way merging) to reduce the I/Os further. Apart from carefully tuning the parameters, they have incorporated many optimization techniques to save constant factors in the total number of I/Os.

The Euler-tour-based variant of the implementation of MM_BFS also benefits from various heuristics. The portion of the hot pool H that fits in internal memory is kept as a multi-map hash table. This reduces the internal memory computation significantly as the hot pool need not be scanned completely for each level.

After the Euler-tour-based clustering, any set of nodes laid out contiguously on the disk is also likely to be close together in terms of BFS levels, irrespective of the cluster boundaries. As such, the other adjacency lists brought into the main memory while loading a cluster may also be useful soon after. Therefore, if the hot pool fits in main memory, these extra nodes are also cached in the main memory. Note that this does not affect the worse case asymptotic I/O complexity of the algorithm as scanning the hot pool as well as the cached adjacency list does not cause any I/Os (as it all fits in the main memory).

6.4 EM BFS in Practice

Ajwani et al.'s implementation [4,5] of the external memory BFS algorithms [49, 53] has made BFS viable for massive graphs on a low cost machine. On many different classes of graphs of about one billion edges, this implementation computes a BFS level decomposition in a few *hours*, which would have taken the traditional RAM model BFS algorithm [25] several *months*. For instance, the BFS level decomposition of a web-crawl based graph of around 130 million nodes and 1.4 billion edges can be computed in less than 5 hours on a machine with a 2.0 GHz Intel Xeon processor, one GB of RAM, 512 KB cache and 250 GB Seagate Barracuda hard disk. Disk parallelism further alleviates the I/O bottleneck. For instance, BFS on the web-crawl based graph takes only 3 hours with 4 disks.

On low diameter graphs, MR_BFS performs quite well as it requires $O(\mathrm{sort}(m))$ I/Os for computing a single BFS level. For random graphs with diameter $O(\log n)$, MR_BFS only requires $O(\mathrm{sort}(m) \cdot \log n)$ I/Os. On large diameter graphs like a list randomly distributed on the disk (without explicit information of it being a list), the Euler tour based variant of MM_BFS together with the heuristics performs well. Even on moderate diameter graphs like grid graphs, the Euler tour based variant of MM_BFS is still better. On a 2-dimensional grid with 2^{28} nodes and roughly 2^{29} edges, it takes around 21 hours using a single disk. As such, moderate diameter graphs continue to be a challenge for BFS implementations.

If there is a priori information regarding the diameter of the input graph, it can make the BFS implementations even faster by appropriately modifying the value of μ. External memory algorithms for computing approximate diameter may also help in giving such an information.

7 Single-Source Shortest-Paths

The single-source shortest-paths (SSSP) problem takes as an input a large weighted undirected graph $G(V, E)$ and a source node s and computes the shortest paths $d(s, v)$ for all nodes $v \in V$. They can be computed in $O(n \log n + m)$ in internal memory using Dijkstra's algorithm [31] with a Fibonacci heap-based priority queue [36]. Dijkstra's algorithm relies heavily on the priority queue. It starts by inserting all neighbors x of the source node s into the priority queue with the weight of the edge $\{s, x\}$ $(W(s, x))$ as their key. Thereafter, the algorithm iteratively deletes the minimum element v with priority $d(s, v)$ from the priority queue. It then decreases the key (using the decrease_key operation) of the unvisited out-neighbors w of v to $d(s, v) + W(v, w)$ in the priority queue. The decrease_key operation inserts the node w into the priority queue if it did not exist before. If the node w existed in the priority queue and had a key higher than $d(s, v) + W(v, w)$, this operation reduces its key value to $d(s, v) + W(v, w)$. Otherwise, this operation is ignored.

SSSP suffers from all the three problems in external memory graph traversal: Keeping track of settled nodes (the nodes already deleted from the priority queue so as not to insert them again), unstructured accesses to the adjacency lists, and the lack of an efficient addressable external memory priority queue.

The SSSP algorithm by Kumar and Schwabe [46] (KS_SSSP) uses I/O-efficient tournament trees [46] for priority queue operations. On an I/O-efficient tournament tree with n elements, any sequence of $z \geq B$ delete, delete_min, update operations requires at most $O((z/B) \cdot \log_2 (n/B))$ I/Os. Let us denote an instance of such a tree in undirected SSSP algorithm as PQ.

Note that not marking whether or not a node has been visited before affects the undirected SSSP only in a limited way: For the undirected edge $\{u, v\} \in E$ with $W(u, v) > 0$ and $d(s, u) < d(s, v)$, the node u may be re-inserted into the priority queue PQ and may get extracted again with key $d(s, v) + W(v, u)$. For the correctness of the algorithm, it is sufficient to delete u from the priority queue somewhere between the extraction of v and re-extraction of u. Recall from the analysis of internal memory Dijkstra's algorithm that the key value of extracted elements from priority queue continue to increase over the course of the algorithm. Deleting u from PQ when the key value has reached $d(s, u) + W(u, v)$

is sufficient for the correctness of the algorithm because when v is extracted the key value is $d(s,v)(\leq d(s,u)+W(u,v))$ and if u were to be re-extracted, its key value would then be $d(s,v)+W(u,v) > d(s,u)+W(u,v)$. Thus this deletion will happen between the extraction of v and the re-extraction of u and the algorithm will behave correctly.

In order to "remember" deleting u from PQ at "the right time", KS_SSSP uses another external memory priority queue PQ^*. It then checks the smallest elements (u_1,k_1), (u_2,k_2) of PQ and PQ^*. If $k_1 < k_2$, it extracts u_1 from PQ and for all out-neighbors v of u_1, it decreases the key of v in PQ to $d(s,u_1)+W(u_1,v)$ and inserts a new signal $(u_1, d(s,u_1)+W(u_1,v))$ in PQ^*. Otherwise, a delete(u_2) is performed on PQ. The process is repeated till PQ is empty.

Since the $O(m)$ operations on PQ and PQ^* require $O((m/B)\cdot\log_2(m/B))$ I/Os, the total I/O complexity of this algorithm is $O(n+(m/B)\cdot\log_2(m/B))$. Once again, the $O(n)$ term comes from unstructured accesses to adjacency lists.

7.1 Meyer and Zeh Algorithm

As regards resolving the problem of unstructured accesses to adjacency lists, Meyer and Zeh [51] proposed an algorithm MZ_SSSP that has a preprocessing phase where the adjacency lists are re-arranged on the disk. Unlike BFS where the edges are all unweighted, MZ_SSSP distinguishes between edges with different weights and separates the edges into categories based on their weights.

The category-i edges have weight between 2^{i-1} and 2^i. Let $G_0,\ldots,G_r$ be a sequence of graphs defined as $G_0=(V,\emptyset)$ and, for $1\leq i\leq r$, $G_i=(V,E_i)$ with $E_i=\{e\in E : e \text{ is in category } j\leq i\}$. The category of a cluster C is the smallest integer i such that C is completely contained in a connected component of G_i. The diameter of C is the maximal distance in G between any two nodes in C. For some $\mu\geq 1$, we call a clustering $P=(C_1,\ldots,C_q)$ well-structured if

1. $q=O(n/\mu)$
2. No node v in a category-i cluster C_j has an incident category-k edge (u,v) with $k<i$ and $u\notin C_j$
3. No category-i cluster has diameter greater than $2^i\mu$.

MZ_SSSP computes a well-structured cluster partition of the input graph. This is done by iterating over graphs $G_0,\ldots,G_r$ and computing connected components in each of them. For each G_i, we form the clusters by chopping the Euler tour around a spanning tree of each connected component. While computing clusters of category-i, we ensure that every category-i cluster has diameter at most $2^i\mu$ and that no category-$(i-1)$ component has nodes in two different category-i clusters. The latter invariant means that we have a complete hierarchy of clusters. The edges of the graph are then re-arranged so as the adjacency lists of nodes in a cluster are contiguous on the disk.

The SSSP phase of MZ_SSSP keeps a hierarchy of hot pools $H_0,\ldots,H_r$ and inspects only a subset $H_1,\ldots,H_i$ of these pools in each iteration. It exploits the following observations:

- Relaxing edges with large weights can be delayed. In particular, it suffices to relax a category-i edge incident to a settled node v any time before the first node at distance at least $d(s, v) + 2^{i-1}$ from s is settled.
- An edge in a category-i component cannot be up for relaxation before the first node in the component is about to be settled.

The first observation allows us to store all category-i edges in pool H_i, as long as we guarantee that H_i is scanned at least once between the settling of two nodes whose distances from s differ by at least 2^{i-1}. The second observation is exploited by storing even category-j edges, $j < i$, in pool H_i, as long as we move these edges to lower pools as the time of their relaxation approaches.

In order to eliminate the effect of spurious updates, MZ_SSSP uses a second priority queue, similar to KS_SSSP.

The total I/O complexity of this algorithm is $O(\sqrt{n \cdot \text{scan}(m) \cdot \log W} + MST(n, m))$ I/Os, where W is the ratio between the weights of the heaviest and the lightest edge and $MST(n, m)$ is the number of I/Os required to compute a minimum spanning tree of a graph with n nodes and m edges.

Meyer and Zeh [52] extended this framework to handle the case of unbounded edge-weights. Their algorithm for SSSP with unbounded edge-weights requires $O(\sqrt{n \cdot \text{scan}(m)} \cdot \log n + MST(n, m))$ I/Os. In contrast, the best external memory directed SSSP algorithm [23] requires $O(n + \lceil \frac{n}{M} \rceil \cdot \text{scan}(n + m) + \text{sort}(m))$.

Brodal et al. [16] showed that SSSP on an undirected graph can be computed in $O(n + \text{sort}(m))$ I/Os with a cache-oblivious algorithm relying on a cache-oblivious bucket heap for priority queue operations. Allulli et al. [6] gave a cache-oblivious SSSP algorithm improving the upper bound to $O(\sqrt{n \cdot \text{scan}(m) \cdot \log W} + \text{scan}(m) \cdot \log n + \text{sort}(m) + MST(n, m))$, where W is the ratio between the smallest and the largest edge weight and $MST(n, m)$ is the I/O complexity of the cache-oblivious algorithm computing a minimum spanning tree of a n node and m edge graph.

7.2 Engineering EM SSSP

Recently, some external memory SSSP approaches (similar in nature to the one proposed in [46]) have been implemented [22, 57] and tested on graphs of up to 6 million nodes. However, in order to go external and still not produce huge running times for larger graphs, these implementations restrict the main memory size to rather unrealistic 4 to 16 MB.

Meyer and Osipov engineered a practical I/O-efficient single-source shortest-paths algorithm on general undirected graphs where the ratio between the largest and the smallest edge weight is reasonably bounded. Their implementation is semi-external as it assumes that the main memory is big enough to keep some constant bits of information per node. This assumption allows them to use a bit vector of size n kept in the internal memory for remembering settled nodes.

In order to get around the lack of optimal decrease_key operation in current external memory priority queues, it allows up to d_v (degree of node v) many entries for a node v in the priority queue at the same time and when extracting them, it discards all but the first one with the help of the bit vector. As regards accessing the adjacency lists in an unstructured way, they do a preprocessing similar to the Euler tour based variant of MM_BFS (i.e., without considering the edge weights at all) to form clusters of nodes. For integer edge weights from $\{1, \ldots, W\}$ and $k = \log_2 W$, the algorithm keeps k "hot pools" where the i-th pool is reserved for edges of weight between 2^{i-1} and $2^i - 1$. It loads the adjacency lists of all nodes in a cluster into these "hot pools" as soon as the first node in the cluster is settled.

In order to relax the edges incident to settled nodes, the hot pools are scanned and all relevant edges are relaxed. The algorithm crucially relies on the fact that the relaxation of large weight edges can be delayed because for such an edge (even assuming that it is in the shortest path), it takes some time before the other incident node needs to be settled. The hot pools containing higher weight edges are thus touched less frequently than the pools containing short edges.

Similar to the implementation of MM_BFS, it partially maintain the pool in the internal memory hash table for efficient dictionary look up rather than computationally quite expensive scanning of all hot pool edges. The memory can be shared between "hot pools" either uniformly or in an exponentially decreasing way. The latter makes sense as the hot pools with lighter edges are scanned more often. When the clusters are small enough, the algorithm caches all neighboring clusters that are anyway loaded into the main memory while reading B elements from the disk.

For random edge weights uniformly distributed in $[1, \ldots, W]$, the expected number of I/Os incurred by this algorithm is $O(\sqrt{(n \cdot m \cdot \log W)/B} + MST(n, m))$, the same as that for MZ_SSSP.

Their pipelined implementation makes extensive use of STXXL algorithms and data structures such as sorting and priority queues.

7.3 SSSP in Practice

As predicted theoretically, this SSSP approach is acceptable on graphs with uniformly distributed edge weights. For random graphs (2^{28} nodes and 2^{30} edges) with uniformly random weights in $[1, \ldots, 2^{32}]$, it requires around 40 hours to compute SSSP (with 1 GB RAM). On a US road network graph with around 24 million nodes and around 29 million edges, it requires only around half an hour for computing SSSP, even when the node labels are randomly permuted before. On many difficult graph classes for BFS, the running time of this SSSP approach is within a factor of two to the BFS implementation [49].

The final performance of this algorithm has been shown to be significantly dependent on the quality of the spanning tree and the way space is allocated in the main memory among different "hot pools".

8 All-Pair Shortest-Paths

The I/O efficient All-Pair Shortest-Paths (APSP) algorithm [11] on undirected graphs uses KS_SSSP as a building block. However, it solves all n underlying

SSSP problems "concurrently". This requires n priority queue pairs (PQ_i, PQ_i^*), $0 \leq i \leq n$, one for each SSSP problem.

The I/O-efficient tournament tree (I/O-TT) as used by KS_SSSP requires $O((z/B) \cdot \log(n/B))$ I/Os for a sequence of $z \geq B$ delete, delete-min and update operations, provided the internal memory available for the tree M' satisfies $M' = \Theta(B)$. If $M' < B$, it has a worst case bound of $O((z/M') \cdot \log(n/M'))$ I/Os. Since for APSP, we need $2n$ of these priority queues working "in parallel", we need an I/O-efficient multi-tournament-tree (I/O-MTT). An I/O-MTT consists of L independent I/O-TTs each with $M' = \Theta(B/L)$. In other words, the root nodes of all L bundled I/O-TTs can be kept in one block.

The I/O-efficient APSP proceeds in $O(n)$ rounds. In each round, it loads the roots of all its I/O-MTTs and extracts a settled graph node with smallest distance for each pair (PQ_i, PQ_i^*) of I/O-TTs. It sorts these nodes according to their labels and then, with one parallel scan of the graph representation, it collects all relevant adjacency lists. With another sorting step, these adjacency lists are moved back to the I/O-TT pairs of the SSSP problem from which they originated. This is followed by the necessary update operations in each pair of I/O-TTs, which requires another round of cycling through all I/O-MTTs.

Cycling through all I/O-MTTs requires $O(\frac{n}{L})$ I/Os. This sums up to $O(\frac{n^2}{L})$ I/Os over all rounds. On each I/O-TT, we perform $O(m)$ I/O operations, requiring a total of $O(n \cdot \mathrm{scan}(m) \cdot L \cdot \log n)$ I/Os for all I/O-TTs. In each round, we require only $O(\mathrm{sort}(m))$ I/Os to get the adjacency lists of extracted nodes. Thus, over all rounds, we need $O(n \cdot \mathrm{sort}(m))$ I/Os to fetch all requisite adjacency lists. Substituting $L = O(\sqrt{n/(\mathrm{scan}(m) \cdot \log n)})$, we get the total I/O complexity for APSP on undirected graphs to be $O(n \cdot (\sqrt{(n \cdot \mathrm{scan}(m) \cdot \log n)} + \mathrm{sort}(m)))$, provided $m/n \leq B/\log n$.

Chowdhury and Ramachandran [24] improved this result by using a multi-buffer-heap in place of an I/O-MTT. Their cache-oblivious algorithm for weighted undirected graphs requires $O(n \cdot (\sqrt{n \cdot m/B} + (m/B) \cdot \log(m/B)))$ I/Os.

For undirected unweighted graphs, the straightforward way of performing all pair shortest path or all pair BFS is to use MM_BFS on each node of the graph using $O(n \cdot (\sqrt{n \cdot m/B} + \mathrm{sort}(m) \cdot (1 + \log\log(n \cdot B/m))))$ I/Os in total. This bound can be improved by running MM_BFS "in parallel" from all source nodes in the same cluster and considering source clusters one after the other. We initially compute a clustering of the input graph (once and for all) like in MM_BFS. Then we iterate through all clusters. For a particular source cluster C_s, we run MM_BFS for all source nodes $s \in C_s$ "in parallel" using a common hot pool H. Each cluster is brought into the hot pool using random access (and possibly some sequential I/Os) precisely once for each source cluster. However, once brought in the pool, the adjacency list of all nodes in the cluster will have to remain in the pool until all the BFS-trees of the current source cluster C_s reach v. Note that the adjacency lists still stay in the hot pool for $O(\mu)$ rounds as once v is visited in BFS from some node in C_s, it takes $O(\mu)$ more rounds for BFS from all other nodes in C_s to also visit v.

For each of the source cluster, an analysis similar to that of MM_BFS shows an I/O bound of $O(n/\mu + \mu \cdot m/B + \mu \cdot \text{sort}(m))$. Substituting $\mu = O(\sqrt{n \cdot B/(m)})$ and summing up over all the clusters, we get a total I/O complexity of $O(n \cdot \text{sort}(m))$ for AP-BFS.

9 A*

A* [40] is a goal-directed graph traversal strategy that finds the least-cost path from a given source node to a target node. The key difference between Dijkstra's algorithm and A* is that while Dijkstra's algorithm uses a priority queue to extract the node with the minimum distance from the source to visit, A* visits the node with the minimum sum of distance from the source node and the heuristic distance to the target node.

Similar to KS_SSSP, A* can be solved using external memory priority queues in $O(n + m/B \cdot \log_2 (m/B))$ I/Os. For implicit unweighted graphs, a suitably modified version of MR_BFS helps computing A* in $O(\text{sort}(m))$ I/Os [33]. This is because in implicit graphs accessing the adjacency list of a node does not require I/Os to fetch it from the disk, but only internal memory computation to generate it.

External A* has also been extended to work on directed and weighted graphs arising in checking safety properties in model checking domains [41] and liveness properties [32].

9.1 A* in Practice

The practical performance of A* crucially depends on the heuristic estimate of the distance between target node and a given node. This estimate in turn is heavily application-dependent.

Edelkamp et al. [42] engineered the external memory A* for implicit undirected unweighted graphs with a variant of MR_BFS and used it for many different model checking applications. They improved the practical performance of external memory A* for their applications further by the following heuristics:

- Delayed duplicate detection: Unlike MR_BFS, duplicates are not removed till the nodes are actually visited.
- The nodes that have equal value of the sum of distances from the source and the target node are visited in increasing order of their distance from the source node

External A* as incorporated in the External SPIN model checker software was used to detect the optimal path to a deadlock situation in an Optical Telegraph protocol involving 14 stations. This problem required 3 Terabytes of hard disk space (with 3.6 GB RAM) and took around 8 days with 4 instances of Parallel External SPIN running on 4 AMD Opteron dual processor machines with NFS shared hard disk. In model checking applications involving a massive state space, finding such deadlocks can be critical for the correct performance of the protocol and hence, even running-times of weeks are considered acceptable.

10 Traversing Directed Graphs

The internal memory DFS algorithm explores the nodes of a graph in a node-by-node fashion. It maintains a stack of nodes, which is initialized with the source node. It iteratively examines the top node v on the stack, finds the next unexplored edge out of v, and if the edge points to an unvisited node, pushes the new node on the stack. If there are no unexplored edges out of v, it is removed from the stack.

As discussed in Section 4, the key difficulty in externalizing this and other directed graph traversal algorithms is determining whether the next unexplored edge points to an unvisited node, without doing $\Omega(1)$ I/Os per edge. The asymptotically best external memory directed graph traversal algorithms rely on buffered repository trees [19] for removing edges leading to visited nodes.

When a node v is first visited, all its incoming edges (x, v) are kept in a buffered repository tree BRT, keyed by x. For each node v, we keep a priority queue $PQ(v)$. Initially, $PQ(v)$ contains all outgoing edges of node v in the order that DFS will traverse them.

Suppose node u is currently at the top of the stack, and we are looking for an unvisited out-neighbor of u. We extract all edges with key u from BRT. This set of edges (u, w) is such that either node w has been discovered between the current time and the last time u was at the top of the stack, or this is the first time u is at the top of the stack since the start of the algorithm. For each edge (u, w) extracted in this way, we delete w from $PQ(u)$. A delete-min operation on $PQ(u)$ reveals the next edge (u, x) to scan. The deletions maintain the invariant that, prior to each delete-min, $PQ(u)$ contains exactly those out-edges of u that point to its unvisited out-neighbors. Therefore, we can push x onto the stack and iterate, assigning x the next DFS number in the process. If $PQ(u)$ is empty, then u is popped and we iterate.

BFS is similar, except that the stack is replaced by a FIFO queue and rather than using the priority queue, we use the following simple operation: For all out-neighbors w of u such that the edge (u, w) is not in the BRT, we push w at the end of the FIFO queue.

Rearranging the input graph into the adjacency list (or adjacency vector) format requires $O(\text{sort}(m))$ I/Os. The stack and the FIFO queue operations can all be done in $O(\text{scan}(n))$ and $O(\text{scan}(m))$ I/Os, respectively. The m inserts and $2n$ extracts from the buffered repository tree cause $O((n + m/B)\log_2 \frac{n}{B})$ I/Os.

The first (root) block of the priority queue $PQ(v)$ is loaded into internal memory whenever the node v is at the top of the stack. Since there are $O(n)$ stack operations in total, the cost for loading the priority queues into internal memory and performing all the insert, delete and delete-min operations on them is $O(n + \text{sort}(m))$ I/Os.

The resultant I/O complexity of BFS and DFS on directed graphs is thus $O((n + m/B)\log_2 \frac{n}{B} + \text{sort}(m))$ I/Os.

Owing to the $O(n \log_2 \frac{n}{B})$ term in the I/O complexity, these algorithms are considered impractical for general sparse directed graphs. Since real world graphs are usually sparse, it is unlikely that these algorithms will improve the running time significantly as compared to the internal memory traversal algorithms. As such, there has been no engineering attempt for these algorithms.

10.1 Directed DFS in Practice

Sibeyn et al. [61] showed an implementation of semi-external DFS (i.e., computing DFS when $M \geq c \cdot n$ for some small constant c) based on the batched processing framework. Their implementation assumes that the internal memory can contain up to $2n$ edges. It maintains (the edges of) a tentative DFS tree throughout the algorithm in the internal memory and proceeds in rounds. In each round, all the edges of the graph are processed in cyclic order. A round consists of m/n phases and in each phase it loads a batch of n edges and computes the DFS of the (recently loaded n edges and the $n-1$ edges of the tentative DFS tree) $2n-1$ edges in the internal memory. The DFS computation is made faster by the following heuristics [61]:

- Rearrange the tree after every round so as to find the global DFS tree more rapidly. For each node, it visits its children (in the tree) in descending order of their sub-tree sizes. Thus, after rearrangement the leftmost child of any node has more descendants than any other child, thereby heuristically reducing the number of forward (left to right) cross edges.
- Reduce the number of nodes and edges "active" in any round so as to leave more space in the internal memory for loading new edges. Since nodes on the leftmost path are not going to change their place in the tree anymore (unless they are rearranged), they can be marked "passive" and removed from consideration. Furthermore, this implementation marks all nodes u that satisfy the following conditions passive:
    - All nodes on the path from root node to u are already marked passive.
    - There is no edge from any node with smaller pre-order number (in the current tree) to any node with pre-order number equal to or larger than that of u.

Together with these heuristics, the batched processing framework manages to compute DFS on a variety of directed graphs (such as random graphs and 2-dimensional random geometric graphs) with very few (3–10) average accesses per edge (and hence few I/Os). It can compute strongly connected components (using the DFS) of an AT&T call graph with around 9.9 million nodes and 268.4 million edges in around 4 hours on a Pentium III machine with a 1 GHz processor.

11 Conclusion

The design and analysis of external memory graph traversal algorithms has greatly improved the worst case upper bounds for the I/O complexity of many graph traversal problems. Table 1 summarizes the state-of-the-art in external memory graph traversal algorithms on general graphs.

Table 1. I/O complexity of state-of-the-art algorithms (assuming $m \geq n$) for graph traversal problems

Problems	Best known upper bounds	Reference
MST/CC (deterministic on undirected graphs)	$O(\text{sort}(m) \cdot \log\log(n \cdot B/m))$	[9]
MST/CC (randomized on undirected graphs)	$O(\text{sort}(m))$	[1]
List ranking	$O(\text{sort}(m))$	[23]
Euler Tour	$O(\text{sort}(m))$	
BFS (on undirected graphs)	$O(\sqrt{n \cdot \text{scan}(m)} + \text{sort}(m) + ST(n,m))$	[49]
BFS, DFS, SSSP and Topological ordering (on directed graphs)	$O(\min\{n + \lceil n/M \rceil \cdot \text{scan}(m) + \text{sort}(m),$ $(n + \text{scan}(m)) \cdot \log n, m\})$	[19,23,46]
SSSP (on undirected graphs)	$O(\min\{n + \text{scan}(m) \cdot \log(m/B),$ $\sqrt{n \cdot \text{scan}(m)} \cdot \log W + MST(n,m),$ $\sqrt{n \cdot \text{scan}(m)} \cdot \log n + MST(n,m)\})$	[46,51,52]
APSP (on unweighted undirected graphs)	$O(n \cdot \text{sort}(m))$	[11]
APSP (on undirected graphs with non-negative weights)	$O(n \cdot (\sqrt{n \cdot \text{scan}(m)}+$ $\text{scan}(m) \cdot \log(m/B)))$	[24]

Engineering some of these algorithms has extended the limits of the graph size for which a traversal can be computed in “acceptable time”. This in turn means that optimization problems of larger and larger sizes are becoming viable with advances in external memory graph traversal algorithms.

Many of these algorithms are still far from optimal. Similarly, while implementations of these algorithms provide good results on simple (low or high diameter) graph classes, these are still far from satisfactory for the difficult graph classes. More work is required both in designing and engineering these algorithms, particularly for directed graphs, to make traversal on even larger graphs viable.

References

1. Abello, J., Buchsbaum, A., Westbrook, J.: A functional approach to external graph algorithms. Algorithmica 32(3), 437–458 (2002)
2. Abello, J., Pardalos, P., Resende, M.: On maximum clique problems in very large graphs. External Memory Algorithms, AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science 50, 119–130 (1999)
3. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Communications of the ACM 31(9), 1116–1127 (1988)
4. Ajwani, D., Dementiev, R., Meyer, U.: A computational study of external memory bfs algorithms. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 601–610 (2006)

5. Ajwani, D., Meyer, U., Osipov, V.: Improved external memory BFS implementation. In: Proceedings of the workshop on Algorithm Engineering and Experiments (ALENEX), pp. 3–12 (2007)
6. Allulli, L., Lichodzijewski, P., Zeh, N.: A faster cache-oblivious shortest-path algorithm for undirected graphs with bounded edge lengths. In: Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 910–919 (2007)
7. Arge, L.: The Buffer Tree: A new technique for optimal I/O-algorithms. In: Sack, J.-R., Akl, S.G., Dehne, F., Santoro, N. (eds.) WADS 1995. LNCS, vol. 955, pp. 334–345. Springer, Heidelberg (1995)
8. Arge, L., Bender, M.A., Demaine, E.D., Holland-Minkley, B., Munro, J.I.: Cache-oblivious priority queue and graph algorithm applications. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), pp. 268–276 (2002)
9. Arge, L., Brodal, G., Toma, L.: On external-memory MST, SSSP and multi-way planar graph separation. In: Halldórsson, M.M. (ed.) SWAT 2000. LNCS, vol. 1851, pp. 433–447. Springer, Heidelberg (2000)
10. Arge, L., Chase, J.S., Halpin, P., Toma, L., Vitter, J.S., Urban, D., Wickremesinghe, R.: Efficient flow computation on massive grid terrain datasets. Geoinformatica 7(4), 283–313 (2003)
11. Arge, L., Meyer, U., Toma, L.: External memory algorithms for diameter and all-pairs shortest-paths on sparse graphs. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 146–157. Springer, Heidelberg (2004)
12. Arge, L., Procopiuc, O., Vitter, J.: Implementing I/O-efficient data structures using TPIE. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 88–100. Springer, Heidelberg (2002)
13. Boruvka, O.: O jistém problému minimálním. In: Práce, Moravské Prirodovedecké Spolecnosti, pp. 1–58 (1926)
14. Brandes, U., Erlebach, T. (eds.): Network Analysis. LNCS, vol. 3418. Springer, Heidelberg (2005)
15. Brodal, G., Fagerberg, R.: Cache oblivious distribution sweeping. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 426–438. Springer, Heidelberg (2002)
16. Brodal, G., Fagerberg, R., Meyer, U., Zeh, N.: Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 480–492. Springer, Heidelberg (2004)
17. Brodal, G.S., Fagerberg, R.: Funnel heap - a cache oblivious priority queue. In: Bose, P., Morin, P. (eds.) ISAAC 2002. LNCS, vol. 2518, pp. 219–228. Springer, Heidelberg (2002)
18. Brodal, G.S., Fagerberg, R., Vinther, K.: Engineering a cache-oblivious sorting algorithm. In: Proceedings of the sixth workshop on Algorithm Engineering and Experiments (ALENEX), pp. 4–17 (2004)
19. Buchsbaum, A., Goldwasser, M., Venkatasubramanian, S., Westbrook, J.: On external memory graph traversal. In: 11th Annual Symposium on Discrete Algorithms (SODA), pp. 859–860. ACM-SIAM (2000)
20. Chan, T.M.: All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. Algorithmica 50(2), 236–243 (2008)
21. Chazelle, B.: A minimum spanning tree algorithm with inverse-ackermann type complexity. JACM 47(6), 1028–1047 (2000)

22. Chen, M., Chowdhury, R.A., Ramachandran, V., Roche, D.L., Tong, L.: Priority queues and dijkstra's algorithm. Technical report TR-07-54, The University of Texas at Austin, Department of Computer Sciences (2007)
23. Chiang, Y.J., Goodrich, M.T., Grove, E.F., Tamasia, R., Vengroff, D.E., Vitter, J.S.: External memory graph algorithms. In: Proceedings of the 6th annual Symposium on Discrete Algorithms (SODA), pp. 139–149. ACM-SIAM (1995)
24. Chowdhury, R.A., Ramachandran, V.: External-memory exact and approximate all-pairs shortest-paths in undirected graphs. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 735–744 (2005)
25. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. The MIT Press, Cambridge (1989)
26. de Kunder, M.: World wide web size, `http://www.worldwidewebsize.com/`
27. Dementiev, R., Kettner, L., Sanders, P.: Stxxl: Standard Template Library for XXL Data Sets. Technical Report 18, Fakultät für Informatik, University of Karlsruhe (2005)
28. Dementiev, R., Kettner, L., Sanders, P.: STXXL: Standard Template library for XXL data sets. Software: Practice and Experience 38(6), 589–637 (2008)
29. Dementiev, R., Sanders, P., Kettner, L.: STXXL: Standard Template library for XXL data sets. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 640–651. Springer, Heidelberg (2005)
30. Dementiev, R., Sanders, P., Schultes, D., Sibeyn, J.: Engineering an external memory minimum spanning tree algorithm. In: Proceedings of the 3rd International Conference on Theoretical Computer Science (TCS), pp. 195–208. Kluwer, Dordrecht (2004)
31. Dijkstra, E.W.: A note on two problems in connexion with graphs. In Numerische Mathematik 1, 269–271 (1959)
32. Edelkamp, S., Jabbar, S.: Large-scale directed model checking LTL. In: Valmari, A. (ed.) SPIN 2006. LNCS, vol. 3925, pp. 1–18. Springer, Heidelberg (2006)
33. Edelkamp, S., Jabbar, S., Schrödl, S.: External A*. In: Biundo, S., Frühwirth, T., Palm, G. (eds.) KI 2004. LNCS (LNAI), vol. 3238, pp. 226–240. Springer, Heidelberg (2004)
34. Erdős, P., Rényi, A.: On random graphs. Publ. Math. Debrecen 6, 290–297 (1959)
35. Erdős, P., Rényi, A.: On the evolution of random graphs. Magyar Tud. Akad. Mat. Kutato Int. Kozl. 5, 17–61 (1960)
36. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM 34(3), 596–615 (1987)
37. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: 40th Annual Symposium on Foundations of Computer Science, pp. 285–297. IEEE Computer Society Press, Los Alamitos (1999)
38. Galler, B.A., Fisher, M.J.: An improved equivalence algorithm. Communications of the ACM 7(5), 301–303 (1964)
39. Goldberg, A., Werneck, R.: Computing point-to-point shortest paths from external memory. In: Proceedings of the 7th workshop on Algorithms Engineering and Experiments (ALENEX), pp. 26–40 (2005)
40. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics SSC 4(2), 100–107 (1968)
41. Jabbar, S., Edelkamp, S.: I/O efficient directed model checking. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 313–329. Springer, Heidelberg (2005)

42. Jabbar, S., Edelkamp, S.: Parallel external directed model checking with linear I/O. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 237–251. Springer, Heidelberg (2005)
43. Jarník, V.: O jistém problému minimálním [About a certain minimal problem]. Práce Moravské Přírodovědecké Společnosti 6, 57–63 (1930)
44. Karger, D., Klein, P., Tarjan, R.: A randomized linear time algorithm to find minimum spanning trees. Journal of the ACM 42(2), 321–328 (1995)
45. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society 7(1), 48–50 (1956)
46. Kumar, V., Schwabe, E.J.: Improved algorithms and data structures for solving graph problems in external memory. In: Proceedings of the 8th Symposium on Parallel and Distributed Processing, pp. 169–177. IEEE, Los Alamitos (1996)
47. Laura, L., Leonardi, S., Millozzi, S., Meyer, U., Sibeyn, J.F.: Algorithms and experiments for the webgraph. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 703–714. Springer, Heidelberg (2003)
48. Maon, Y., Scheiber, B., Vishkin, U.: Parallel ear decomposition search (EDS) and st-numbering in graphs. Theoretical Computer Science 47, 277–298 (1986)
49. Mehlhorn, K., Meyer, U.: External-memory Breadth-First Search with sublinear I/O. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 723–735. Springer, Heidelberg (2002)
50. Mehlhorn, K., Naher, S.: The LEDA Platform of Combinatorial and Geometric Computing. Cambridge University Press, Cambridge (1999)
51. Meyer, U., Zeh, N.: I/O-efficient undirected shortest paths. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 434–445. Springer, Heidelberg (2003)
52. Meyer, U., Zeh, N.: I/O-efficient undirected shortest paths with unbounded edge lengths. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 540–551. Springer, Heidelberg (2006)
53. Munagala, K., Ranade, A.: I/O-complexity of graph algorithms. In: Proceedings of the 10th Annual Symposium on Discrete Algorithms (SODA), pp. 687–694. ACM-SIAM (1999)
54. Pettie, S.: A new approach to all-pairs shortest paths on real-weighted graphs. Theoretical Computer Science 312(1), 47–74 (2004)
55. Pettie, S., Ramachandran, V.: An optimal minimum spanning tree algorithm. Journal of the ACM 49(1), 16–34 (2002)
56. Prim, R.C.: Shortest connection networks and some generalisations. Bell System Technical Journal 36, 1389–1401 (1957)
57. Sach, B., Clifford, R.: An empirical study of cache-oblivious priority queues and their application to the shortest path problem (2008), `http://arxiv.org/abs/0802.1026v1`
58. Sanders, P.: Random permutations on distributed, external and hierarchical memory. Information Processing Letters 67, 305–309 (1998)
59. Sanders, P., Schultes, D., Vetter, C.: Mobile route planning. In: Halperin, D., Mehlhorn, K. (eds.) Esa 2008. LNCS, vol. 5193, pp. 732–743. Springer, Heidelberg (2008)
60. Sibeyn, J.F.: From parallel to external list ranking, Technical report, Max Planck Institut für Informatik, Saarbrücken, Germany (1997)
61. Sibeyn, J.F., Abello, J., Meyer, U.: Heuristics for semi-external depth first search on directed graphs. In: Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 282–292 (2002)

62. Stepanov, A., Lee, M.: The Standard Template Library. Hewlett Packard Laboratories (1995), http://www.sgi.com/tech/stl/
63. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. Journal of ACM 22(2), 215–225 (1975)
64. Seagate Technology,
http://www.seagate.com/cda/products/discsales/
marketing/detail/0,1081,628,00.html

Minimum Cycle Bases and Their Applications

Franziska Berger[1], Peter Gritzmann[2], and Sven de Vries[3]

[1] Department of Mathematics, Polytechnic Institute of NYU, Six MetroTech Center,
Brooklyn NY 11201, USA
fberger@poly.edu

[2] Zentrum Mathematik, Technische Universität München, 80290 München, Germany
gritzman@ma.tum.de

[3] FB IV, Mathematik, Universität Trier, 54286 Trier, Germany
devries@uni-trier.de

Abstract. Minimum cycle bases of weighted undirected and directed graphs are bases of the cycle space of the (di)graphs with minimum weight. We survey the known polynomial-time algorithms for their construction, explain some of their properties and describe a few important applications.

1 Introduction

Minimum cycle bases of undirected or directed multigraphs are bases of the cycle space of the graphs or digraphs with minimum length or weight. Their intriguing combinatorial properties and their construction have interested researchers for several decades. Since minimum cycle bases have diverse applications (e.g., electric networks, theoretical chemistry and biology, as well as periodic event scheduling), they are also important for practitioners.

After introducing the necessary notation and concepts in Subsections 1.1 and 1.2 and reviewing some fundamental properties of minimum cycle bases in Subsection 1.3, we explain the known algorithms for computing minimum cycle bases in Section 2. Finally, Section 3 is devoted to applications.

1.1 Definitions and Notation

Let $\boldsymbol{G} = (V, E)$ be a directed multigraph with m edges and n vertices. Let $E = \{e_1, \ldots, e_m\}$, and let $w : E \to \mathbb{R}^+$ be a positive weight function on E. A cycle $\boldsymbol{C}$ in $\boldsymbol{G}$ is a subgraph (actually ignoring orientation) of $\boldsymbol{G}$ in which every vertex has even degree (= in-degree + out-degree). We generally neglect vertices of degree zero. $\boldsymbol{C}$ is called *simple* if it is connected and every vertex has degree two. The weight of a cycle $\boldsymbol{C}$ with respect to w is defined as $w(\boldsymbol{C}) := \sum_{e \in \boldsymbol{C}} w(e)$.

We want to associate F-vector spaces (usually for $F \in \{GF(2), \mathbb{Q}\}$) to the cycles of $\boldsymbol{G}$ and to study in particular their w-minimal bases. Towards this, choose a *cyclic order* of the edges in $\boldsymbol{C}$. The entries of the *incidence vector* $b(\boldsymbol{C})$

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 34–49, 2009.

of $\boldsymbol{C}$ are defined as follows with respect to this order, for $i = 1, \ldots, m$.

$$b_i(\boldsymbol{C}) = \begin{cases} -1, & \text{if } e_i \in \boldsymbol{C} \text{ and } e_i \text{ occurs in backward direction} \\ 0, & \text{if } e_i \notin \boldsymbol{C} \\ +1, & \text{if } e_i \in \boldsymbol{C} \text{ and } e_i \text{ occurs in forward direction.} \end{cases}$$

For simple cycles, there are two different orders. These orders lead to incidence vectors of opposite sign; as we are interested mainly in the linear spaces spanned by them, this order is inconsequential. For $F = GF(2)$ the orientation of the edges is ignored; so one could alternatively define the incidence vector as the ordinary binary incidence vector of cycles in the underlying graph.

The F-vector space $\mathcal{C}_F(\boldsymbol{G})$, spanned by the incidence vectors of all cycles in $\boldsymbol{G}$ is called *F-cycle space* of $\boldsymbol{G}$. A basis of $\mathcal{C}_F(\boldsymbol{G})$ is called *F-cycle basis.* The weight of an F-cycle basis is defined as the weight of its binary incidence vectors: $w(\mathcal{B}) = \sum_{\boldsymbol{C} \in \mathcal{B}} \sum_{e: C_e \neq 0} w(e)$. If $w(e) = 1$ for all $e \in E$, we also speak of $w(\mathcal{B})$ as the *length* of $\mathcal{B}$. A *minimum F-cycle basis* has minimum weight among all F-cycle bases. We omit the prefix "F" if F is clear from the context.

The idea to consider different fields F is motivated by the applications, see Section 3, some of which are formulated over $\mathbb{Q}$, whereas the cycle-basis problem was originally studied only for $GF(2)$. Additionally, there are interesting differences among elements of the cycle spaces over different fields (see Fig. 1). However, it is well-known that the dimension of the vector spaces always coincides, see, e.g., [1,2].

Proposition 1. *The dimension of the F-cycle space of a digraph $\boldsymbol{G}$ is equal to $\mu(\boldsymbol{G}) := |E| - |V| + c(\boldsymbol{G})$, where $c(\boldsymbol{G})$ denotes the number of connected components of $\boldsymbol{G}$.*

$\mu := \mu(\boldsymbol{G})$ is called the *cyclomatic number* of $\boldsymbol{G}$. It is also possible to consider the *directed cycle space of $\boldsymbol{G}$*, which is the subspace of $\mathcal{C}_{\mathbb{Q}}(\boldsymbol{G})$ generated by all cycles whose incidence vectors contain only edges of the same orientation. This space coincides with $\mathcal{C}_{\mathbb{Q}}(\boldsymbol{G})$ if and only if $\boldsymbol{G}$ is strongly connected. The algorithms in Section 2 can be executed also in this space, by restricting them to work only on cycles with edges of the same orientation.

1.2 Matroids

The most important property of the F-cycle space of a (di)graph $\boldsymbol{G}$ is that it forms a matroid. We review some standard notions for matroids, cf. [3]. A *matroid* $\mathcal{M}$ is a pair $(E, \mathcal{I})$, where E is a finite ground set and $\mathcal{I}$ is a set of subsets of E satisfying the axioms:

1. $\emptyset \in \mathcal{I}$;
2. if $I \in \mathcal{I}$ and $I' \subseteq I$ then $I' \in \mathcal{I}$; and
3. if $I_1, I_2 \in \mathcal{I}$ and $|I_1| < |I_2|$, then there is an element $e \in I_2 - I_1$ such that $I_1 \cup \{e\} \in \mathcal{I}$.

Subsets of E that belong to $\mathcal{I}$ are called *independent*; all other sets are called *dependent.* Minimal dependent sets of a matroid $\mathcal{M}$ are called *circuits.* A maximal

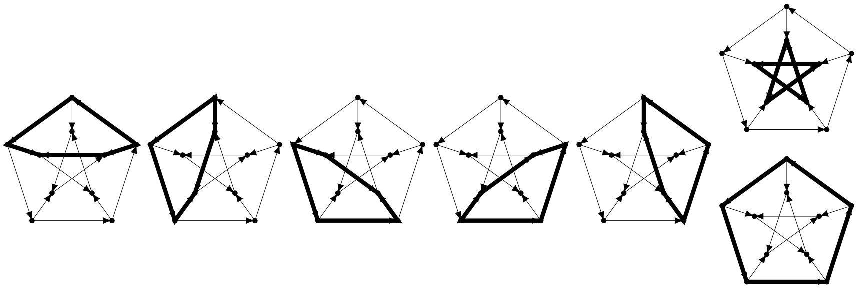


Fig. 1. Cycle bases (in bold) of an orientation of the Petersen graph (with unit edge-weights); the cycle spaces over $\mathbb{Q}$ and $GF(2)$ have dimension $\mu = 15 - 10 + 1 = 6$. The top six cycles clearly form a $\mathbb{Q}$-basis. They are not a $GF(2)$ basis, since every edge is covered twice: thus, the cycles are linearly dependent. The lower six cycles form a cycle basis over $GF(2)$ and therefore over $\mathbb{Q}$. As the Petersen graph has girth 5 and as all cycles in the bases above have length 5, the cycle bases are minimal.

independent set is called a *basis* of the matroid, and all bases have the same cardinality.

Basis exchange is a very important property of matroids: If B_1, B_2 are bases of a matroid and $x \in B_1 \setminus B_2$, then there exists $y \in B_2 \setminus B_1$ such that $(B_1 \setminus \{x\}) \cup \{y\}$ is again a basis. Also, if B is a basis and $e \in E \setminus B$ then $B \cup \{e\}$ contains a unique circuit C; furthermore, $(B \cup \{e\}) \setminus \{f\}$ is a basis for any $f \in C$.

Matroids are further characterized by the property that for any additive weighting $w\colon E \mapsto \mathbb{R}$, the greedy algorithm finds an optimal (maximum-weight) basis.

The F-cycle space of $\boldsymbol{G}$ has the structure of a matroid with sets of F-linearly independent cycles as independent sets. F-cycle bases correspond to bases of this matroid. Thus, cycles in an F-cycle basis may be replaced by certain other cycles without destroying linear independence. More precisely, let $\mathcal{B}$ be a cycle basis and $\boldsymbol{C} \in \mathcal{B}$ a cycle. Then $\boldsymbol{C}$ can be replaced by any linear combination of cycles from the basis in which $\boldsymbol{C}$ has a non-vanishing coefficient.

Because of the basis-exchange property of matroids, every F-cycle basis can be obtained from a given F-cycle basis by a series of such replacements. For minimum F-cycle bases, this implies the following lemma.

Lemma 1. *Let $\mathcal{B}$ be an F-cycle basis. If no cycle $\boldsymbol{C}$ in $\mathcal{B}$ can be exchanged for a shorter cycle $\boldsymbol{D} \notin \mathcal{B}$, then $\mathcal{B}$ is a minimum F-cycle basis.*

Consequently, if the cycles of two minimum F-cycle bases are ordered by increasing weight, their *(sorted) weight vectors*, $\vec{w} := (w(\boldsymbol{C}_1), \ldots, w(\boldsymbol{C}_\mu))$, coincide.

1.3 Basic Properties of Cycle Bases

There are several classes of special cycle bases which are of practical interest. Most important are the *fundamental tree bases* that are constructed from a spanning tree or—if $\boldsymbol{G}$ is not connected—from a spanning forest T of $\boldsymbol{G}$ by adding edges to T. Any non-tree edge e forms a unique *fundamental cycle* $\boldsymbol{F}(e)$ with some edges of T. The set of such cycles has cardinality μ and their linear independence is obvious since every non-tree edge is contained in exactly one cycle. Hence, fundamental tree bases are F-cycle bases.

A fundamental tree basis $\mathcal{B}_T$ can be computed in time $O(mn)$, more specifically in time $O(\sum_{i=1}^{\mu} |\boldsymbol{F}_i(e)|)$, where $\boldsymbol{F}_i(e), i = 1, \ldots, \mu$, are the fundamental cycles [4]. Fundamental tree bases are not necessarily minimal among all cycle bases. Moreover, examples show that a minimum cycle basis need not be a fundamental tree basis [5,6]. See Subsection 2.5 for further discussion.

We may assume that $\boldsymbol{G}$ is simple and two-connected. If it is not, a preprocessing step can be added which takes care of multiple edges and loops and then the 2-blocks can be treated independently, see [7].

Every $GF(2)$-cycle basis is also a $\mathbb{Q}$-cycle basis. The converse is not true, see Fig. 1, and there are examples of graphs where a minimum $\mathbb{Q}$-cycle basis is strictly smaller than a minimum $GF(2)$-cycle basis, see [8,9].

Minimum F-cycle bases have important properties which are used in the algorithms for their construction. For instance, they have to contain certain cycles.

Lemma 2 ([10]). *Let $e \in E$ be an edge of a graph $\boldsymbol{G}$ through which there exists a shortest cycle $\boldsymbol{C}(e)$ with respect to w. Then there is a minimum F-cycle basis $\mathcal{B}$ containing $\boldsymbol{C}(e)$. Moreover, every minimum F-cycle basis must contain some shortest cycle through e.*

In general, the set $\{\boldsymbol{C}(e) : \boldsymbol{C}(e)$ is a shortest cycle through $e \in E\}$ does not span $\mathcal{C}_F(\boldsymbol{G})$, however. The next lemma shows that minimum F-cycle bases contain shortest paths between any two vertices.

Lemma 3 ([11,12]). *Let $v, w \in V$ and $\mathcal{B}$ be an F-cycle basis of $\boldsymbol{G}$. Let P be a shortest path from v to w. Then any cycle $\boldsymbol{C}$ of $\mathcal{B}$ that contains v and w can be exchanged for either a cycle that includes P or a cycle that excludes v or w.*

Lemma 4 ([12]). *Let $\boldsymbol{C}$ be a cycle in a minimum F-cycle basis, and let $u \in \boldsymbol{C}$ be an arbitrary vertex. Then there is an edge $(v, w) \in \boldsymbol{C}$ such that $\boldsymbol{C}$ consists of a shortest u–v path, a shortest u–w path and (v, w).*

Cycles that are contained in at least one minimum cycle basis of $\boldsymbol{G}$ are called *relevant cycles* [11,13] (defined there only for $GF(2)$). Vismara [13] showed the following result for $GF(2)$; it is valid for arbitrary fields.

Lemma 5. *Let $\boldsymbol{G}$ be a graph. A cycle $\boldsymbol{C} \in \mathcal{C}(\boldsymbol{G})$ is relevant if and only if there do not exist simple cycles, $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k$, with the property that $\boldsymbol{C} = \boldsymbol{C}_1 + \cdots + \boldsymbol{C}_k$ and $w(\boldsymbol{C}_i) < w(\boldsymbol{C})$ for all $i = 1, \ldots, k$.*

For $GF(2)$, the addition of two cycles $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ corresponds to the symmetric difference

$$\boldsymbol{C}_1 + \boldsymbol{C}_2 = \boldsymbol{C}_1 \oplus \boldsymbol{C}_2 = (E(\boldsymbol{C}_1) \cup E(\boldsymbol{C}_2)) \setminus E(\boldsymbol{C}_1 \cap \boldsymbol{C}_2)$$

of the underlying edge sets.

2 Algorithms for Computing Minimum Cycle Bases

Some applications require only $GF(2)$-cycle bases while others (such as the electric networks in Subsection 3.1) can at least utilize $GF(2)$-cycle bases, although their structure is over $\mathbb{Q}$. Therefore, we will describe first the $GF(2)$-case in Subsections 2.1–2.3 and then mention the necessary modifications for $\mathbb{Q}$ in Subsection 2.4.

Due to the matroid structure of $\mathcal{C}_F(\boldsymbol{G})$, all known algorithms are based on some sort of greedy argument. However, since the number of simple cycles in a graph can be exponential in n, direct application of the greedy algorithm would be too costly. There are two main approaches to deal with this issue.

The first polynomial algorithm, due to Horton [12], addresses the problem by constructing an $O(mn)$-sized set of cycles that is guaranteed to contain a minimum cycle basis.

The second methodological approach, by de Pina [14] and independently by Berger et al. [7], contains an oracle that creates a cycle which—depending on the point of view—replaces a long cycle in a given basis or, is added to a set of cycles that is already part of a minimum cycle basis.

2.1 Horton's Algorithm

Let us start with Horton's Algorithm 1. It follows from Lemmas 3 and 4 that the cycles of the form $\boldsymbol{C}(x, y, z) = P(z, x) + (x, y) + P(y, z)$ for every triple $\{x, y, z\}$ with $(x, y) \in E$ and $z \in V$, where $P(z, x)$ denotes an *arbitrary* shortest z–x path, contain a minimum cycle basis.

A minimum cycle basis is then extracted from this set by means of the greedy algorithm. Linear independence of the cycles is checked by Gaussian elimination on the corresponding cycle-edge incidence matrix.

```
Input: Two-connected simple edge-weighted digraph $\boldsymbol{G}$
Output: Minimum $GF(2)$-cycle basis $\mathcal{B}$
1. For all $x, y \in G$, find shortest paths, $P(x, y)$, between $x$ and $y$.
2. For all triples $(z, x, y)$ with $e = \{x, y\} \in E$ construct the cycle, $\boldsymbol{C}(z, e) := P(z, x) + e + P(y, z)$, if it is simple.
3. Order all cycles, $\boldsymbol{C}(z, e)$, by increasing weight.
4. Extract greedily a cycle basis, $\mathcal{B}$, from this set by checking linear independence with Gaussian elimination.
```

Algorithm 1. Horton's algorithm to construct a minimum cycle basis

Since the number of candidate cycles is $O(mn)$, a direct implementation has running time $O(m^3 n)$.

2.2 An Exchange Algorithm

The algorithm begins with a fundamental tree basis and successively exchanges cycles for smaller ones if possible. The basic scheme is given in Algorithm 2.

```
Input: Two-connected simple edge-weighted digraph $\boldsymbol{G}$
Output: Minimum $GF(2)$-cycle basis $\mathcal{B}$
1. Construct a fundamental tree basis $\mathcal{B}_0 = \{\boldsymbol{F}_1, \ldots, \boldsymbol{F}_\mu\}$.
2. for $i = 1$ to $\mu$ do
3.     Find a shortest cycle $\boldsymbol{C}_i$ that is linearly independent of $\mathcal{B}_{i-1} \setminus \{\boldsymbol{F}_i\}$.
4.     if $w(\boldsymbol{C}_i) < w(\boldsymbol{F}_i)$ then
5.         $\mathcal{B}_i := (\mathcal{B}_{i-1} \setminus \{\boldsymbol{F}_i\}) \cup \{\boldsymbol{C}_i\}$
6.     end if
7. end for
8. Output $\mathcal{B} := \mathcal{B}_\mu$
```

Algorithm 2. Basic exchange scheme

Of course, the crucial part is the oracle in Line 3 which selects a new cycle $\boldsymbol{C}_i$ in each iteration of the algorithm.

Let $\mathcal{B}$ be a cycle basis. The cycle-edge incidence matrix $A \in GF(2)^{\mu \times m}$ of $\mathcal{B}$ has as rows the (incidence vectors of the) cycles in $\mathcal{B}$. We denote the rows of A as well as the cycles in $\mathcal{B}$ by $\boldsymbol{C}_i, i = 1, \ldots, \mu$.

When looking at linear (in)dependence of cycles, it suffices to restrict attention to the entries corresponding to the non-tree-edges of a spanning tree of the graph: a set of cycles is linearly independent if and only if the submatrix of A corresponding to the non-tree-edges has full rank. Henceforth we will assume that we have fixed a spanning tree for this purpose.

We will now describe Line 3 of Algorithm 2 for a fixed step i. Let U be the inverse matrix of A (restricted to the non-tree edges) and let u_i be the i-th column of U padded with $m - \mu$ zeros (in place of the tree-edges). Denote by $A^{(-i)}$ the matrix that results from A after removing the i'th row $\boldsymbol{C}_i$. Clearly u_i is in the kernel of $A^{(-i)}$. The vector u_i can be computed using standard matrix operations. Alternatively, it is also possible to determine the entire matrix U in each step; this leads to a simple update possibility for step $i + 1$, see [14].

The following trivial remark gives an easy-to-check criterion for whether a cycle $\boldsymbol{C}$ can replace a basis cycle $\boldsymbol{C}_i \in \mathcal{B}$, using u_i:

Remark 1. Let $\mathcal{B}$ be a cycle basis and let $\boldsymbol{C} \notin \mathcal{B}$ be a cycle. $\boldsymbol{C}$ is linearly independent of the row-space of $A^{(-i)}$ if and only if $\langle \boldsymbol{C}, u^i \rangle \neq 0$ holds with the vector u^i defined above.

The vectors $u^i, i = 1, \ldots, \mu$, have the property that $\langle \boldsymbol{C}_j, u^i \rangle = 0$, for $j \neq i$. According to the criterion of Remark 1, it suffices to compute a shortest cycle $\boldsymbol{C}$ for which $\langle \boldsymbol{C}, u^i \rangle = 1$ in step i of the algorithm. This is achieved by computing a shortest path in an auxiliary undirected graph $\boldsymbol{G_u}$. For $u \in \{0, 1\}^E$ the graph $\boldsymbol{G_u}$ is constructed from $\boldsymbol{G}$ as follows (for an example see Figure 2): The vertex set of $\boldsymbol{G_u}$ is $V \times GF(2)$. For each edge $e = \{x, y\}$ in $\boldsymbol{G}$ add the two edges $\{(x, 0), (y, 0 \oplus u_e)\}$ and $\{(x, 1), (y, 1 \oplus u_e)\}$ to $\boldsymbol{G_u}$. Define a weight function $w_u : \boldsymbol{G_u} \to \mathbb{R}^+$ by assigning each edge the weight $w(e)$ of the corresponding edge e in $\boldsymbol{G}$ it comes from. $\boldsymbol{G_u}$ has $2n$ vertices and $2m$ edges. For a vertex v of $\boldsymbol{G}$ and a simple

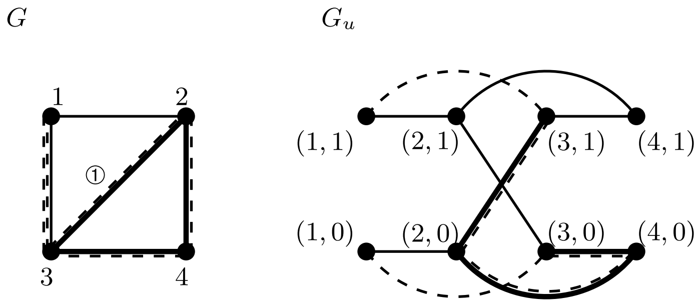


Fig. 2. The dashed (1, 0)–(1, 1) path in $\boldsymbol{G_u}$ does not correspond to a cycle in $\boldsymbol{G}$. However, it contains the bold (3, 0)–(3, 1) sub-path corresponding to cycle $\{2, 3, 4\}$. Here, u contains only one nonzero entry, namely for edge 2–3, displayed as ① in $\boldsymbol{G}$.

$(v, 0)$–$(v, 1)$ path $\boldsymbol{P_v}$ in $\boldsymbol{G_u}$ let $\boldsymbol{W(P_v)}$ denote the closed walk in $\boldsymbol{G}$ obtained by replacing each vertex (x, u) of $\boldsymbol{P_v}$ by x. For the next observation see [7].

Lemma 6. *Let v be a vertex of $\boldsymbol{G}$, and let $\boldsymbol{P_v}$ be a simple $(v, 0)$–$(v, 1)$ path in G_u. Then the closed walk $\boldsymbol{W(P_v)}$ contains a simple cycle $\boldsymbol{C}$ in $\boldsymbol{G}$ with $\langle \boldsymbol{C}, u\rangle = 1$.*

It is not difficult to show that among all paths $\boldsymbol{P_v}$, a shortest one always corresponds to a simple cycle $\boldsymbol{C} = \boldsymbol{W(P_v)}$ in $\boldsymbol{G}$. Hence, to find a shortest cycle $\boldsymbol{C}$ with $\langle \boldsymbol{C}, u\rangle = 1$, one computes a shortest $(v, 0)$–$(v, 1)$ path in $\boldsymbol{G_u}$ for each v and retains only a shortest one.

Theorem 1 ([7]). *Algorithm 2 computes a minimum cycle basis of $\boldsymbol{G}$.*

A direct implementation runs in time $O(\max\{m^3, mn^2 \log n\})$.

2.3 Speed-Ups

There are two major bottlenecks in both algorithms. On the one hand there is a large number of shortest-path computations (n^2 and μn of them, respectively), and on the other hand, for the linear independence tests or the computation of the vectors u^i, Gaussian elimination on an $O(mn) \times \mu(\boldsymbol{G})$ or a $\mu(\boldsymbol{G}) \times \mu(\boldsymbol{G})$ matrix, respectively, is needed.

There are ways to improve on both. Golynski and Horton [15] use a recursive procedure to check linear independence of the cycle-edge incidence matrix of the candidate cycles: the matrix is partitioned into blocks and the pivot operations are performed block-wise. This allows the use of fast matrix multiplication and accelerates the algorithm. Consequently, the performance improves from $O(m^3 n)$ to $O(m^\omega n)$, where ω denotes the matrix multiplication constant (it is presently known, that $\omega < 2.376$).

Kavitha, Mehlhorn, Michail, and Paluch [16] found a similar way to apply fast matrix multiplication and a recursive computation of the vectors u^i with delayed updates to Algorithm 2, which reduces its running time from $O(m^3 + mn^2 \log n)$ to $O(m^2 n + mn^2 \log n)$; for dense graphs, this is an improvement.

Finally, Mehlhorn and Michail [17] have recently improved the running time to $O(m^2 n/\log n + mn^2)$. The idea here is that instead of solving n shortest path problems in each step of Algorithm 2 for constructing the next cycle, one may obtain that cycle by searching through a suitable set of candidate cycles, for which the linear products $\langle \boldsymbol{C}, u^i \rangle$ are computed efficiently from a shortest path tree structure. This approach only requires $O(n)$ shortest path computations. The candidate cycles are a subset of Horton's candidate set from Subsection 2.1.

2.4 The Case of $\mathbb{Q}$

Horton's algorithm from Subsection 2.1 can be used without modification also for computing a $\mathbb{Q}$-minimum cycle basis. Note however, that for Gaussian elimination over $\mathbb{Q}$, it is necessary to pay for the calculations on rational numbers with up to $O(m \log m)$ bits by an additional factor of $\tilde{O}(m)$ (which means $O(m \log^k(m))$ for some k) in asymptotic running time, representing the cost of each arithmetic operation. Hence the algorithm takes time $\tilde{O}(m^4 n)$ if implemented directly. With the divide and conquer strategy sketched in Subsection 2.3, this may be reduced to $\tilde{O}(m^{\omega+1} n)$, see [9].

Algorithm 2 can also be adapted to work over $\mathbb{Q}$, see [18]. First, it has to be determined how to compute the vectors u^i from Remark 1. Kavitha and Mehlhorn show that such vectors with size bounded by $||u^i|| \leq i^{i/2}$ can be computed efficiently. Second, one has to construct a shortest cycle $\boldsymbol{C}$ with $\langle \boldsymbol{C}, u^i \rangle \neq 0$ in each step. This can be done by working in $GF(p)$, for a set of small primes p and applying the Chinese remainder theorem. The shortest path calculation is done in a graph with p levels, but otherwise analogously. The above mentioned block-updates lead to a running time of $O(m^3 n + m^2 n^2 \log n)$. Finally, the variant proposed in [17] gives a running time of $O(m^3 n)$.

2.5 Other Algorithms and Variants

In many cases, it is not required that the F-cycle basis be minimal. Cycle bases of low but not necessarily minimal weight can be computed more quickly. Several different heuristic algorithms exist, see [6,12,19,20,21] and the references therein.

Algorithms for special graphs are also widely discussed, but references will be omitted here. We only mention [22] for planar graphs with running time $O(n^2 \log n + m)$. Approximation algorithms are proposed in [16,23,17], with a constant approximation factor. Randomized algorithms are considered in [24,17].

Cycle bases with additional properties are also of interest. For a discussion of different classes, see, e.g., [20,25]. The problem to find a minimum cycle basis among all fundamental tree bases is APX-hard [26]. The same holds for so-called weakly fundamental cycle bases [27]. However, the existence of an unweighted fundamental tree basis with a length of $O(n^2)$ has been proved recently, see [28].

3 Applications

Cycle bases appear as tools in several areas of engineering. We limit ourselves to three which reflect their main functions. These can be summarized as follows:

- The cycles of the basis are explicitly needed; the basis need not be minimal, but a shorter cycle basis may speed up an ensuing algorithm.
- The cycles are needed explicitly; it is essential that the basis be minimal.
- The cycle basis itself is not needed explicitly; but information is derived from it. It is necessary that the basis is minimal.

3.1 Electric Networks

Cycle bases can be used to model the second Kirchhoff law in the design of electric networks. Electric circuits consist of interconnected network elements whose properties can be described by non-linear differential equations in terms of currents and voltages at time t.

The structure of the interconnection is given by a directed unweighted graph, $\boldsymbol{G} = (V, E)$ whose edges correspond to network elements. Two edges meet in a vertex if the corresponding elements are joined by a wire. More complicated network elements with several ports can be represented by an equivalent circuit diagram consisting of simple circuit elements [29]. The direction of an edge represents the voltage drop with respect to a specified ground node.

In addition to the properties of the circuit elements, the two time-independent Kirchhoff laws govern the behavior of the network:

Current law: The sum of the currents across any edge cut of the network is zero at all times.

Voltage law: The sum of the voltages along any mesh (directed cycle) in the network is zero at all times. If the matrix A is the cycle-edge incidence matrix of a $\mathbb{Q}$-cycle basis, this law can be written as $AU = 0$, where $U \in \mathbb{R}^{|E|}$ denotes the unknown edge voltages (at time t).

The combination of the Kirchhoff laws and the differential equations which describe the circuit elements is called the *network equation*. It completely describes the network and can be used to determine the unknown voltages and currents during the design process. There are different formulations of the network equation; the sparse tableau analysis [30] and the extended modified nodal analysis model, see, e.g., [31] both contain explicitly the equations modeling the Kirchhoff voltage law.

Since the graphs of electric networks can be very large, subsequent computations such as a generic solvability check and error detection algorithms are influenced by the length of the cycle basis used. A short or minimum $\mathbb{Q}$-cycle basis is therefore desirable, in particular if the computations need to be done repeatedly. In practice, short $\mathbb{Q}$-cycle bases will be used rather than minimal ones to improve the running time.

3.2 Chemistry and Biology

For the investigation of functional properties of a chemical compound, it is of interest to study the cycles (or rings) of the molecular graph. The goal is to determine how the ring structure of a molecule influences its chemical and physical properties. One of the most commonly used sets of rings is a minimum $GF(2)$-cycle basis of the graph (called Smallest Set of Smallest Rings, see, e.g., [32]). It is also possible to use the set of relevant cycles instead, for instance.

In this case, a minimum $GF(2)$-cycle basis needs to be explicitly determined. For algorithms which compute all relevant cycles, see [13,33,34].

A different application is the use of a minimum $GF(2)$-cycle basis of the molecular graph for the derivation of graph invariants: Structural information about molecules is available in molecular databases. Most databases offer search and retrieval functionality for molecules also by structure, i.e., the user draws or uploads a molecular graph $\boldsymbol{G}$, and then the database is searched for molecules whose structure is identical to $\boldsymbol{G}$ or which contain substructures identical to $\boldsymbol{G}$.

This identification requires solving the notorious (sub-)graph isomorphism problem for $\boldsymbol{G}$ for each molecule in the database. A common strategy to avoid solving many graph isomorphism problems exactly is to eliminate as many candidate structures as possible by comparing *graph invariants* known in this context as *molecular descriptors*. The most popular descriptors involve the number of atoms and bonds (vertices and edges) and the different atom types. Graphs from the database whose invariants do not coincide with those of $\boldsymbol{G}$ (or do not allow a substructure isomorphic to $\boldsymbol{G}$) are immediately rejected.

As a consequence of the matroid property and Lemma 1, the weight of a minimum cycle basis as well as the weight vector $\vec{w}(\boldsymbol{G})$ are graph invariants. In [34], other invariants are considered which, taken together, provide a strong description of the structure of the cycle space.

The idea is to identify the 'functionality' of a relevant cycle in a minimum $GF(2)$-cycle basis, i.e., stated informally, to characterize the part of the cycle space it spans. Cycles with the same function can in principle be exchanged for each other without violating linear independence; this redundancy defines an equivalence relation, the *interchangeability relation* $\sim_\kappa$, [33], that can be characterized as follows.

Lemma 7. *Let $\boldsymbol{C}, \boldsymbol{C}'$ be two relevant cycles of weight κ. Then $\boldsymbol{C} \sim_\kappa \boldsymbol{C}'$ if and only if there is a representation $\boldsymbol{C} = \boldsymbol{C}' \oplus \bigoplus_{\boldsymbol{D} \in \mathcal{I}} \boldsymbol{D}$, where $\{\boldsymbol{C}'\} \cup \mathcal{I}$ is a (linearly) independent subset of the relevant cycles of weight smaller than or equal to κ.*

This equivalence relation can also be interpreted as a refined form of the connectivity concept in the cycle matroid of the graph. The $\sim_\kappa$ equivalence classes are called *interchangeability classes.*

If a graph contains a unique shortest cycle $\boldsymbol{C}(e)$ for some edge e, this cycle is necessarily contained in every minimum cycle basis, and it cannot be replaced by any other cycle. Hence it forms its own equivalence class with respect to $\sim_\kappa$ where $\kappa = w(\boldsymbol{C})$. On the other hand, there often exist several relevant cycles which are $\sim_\kappa$ interchangeable for $\kappa = w(\boldsymbol{C})$. Minimum cycle bases of unweighted complete graphs, for instance, have only one interchangeability class: suitable $\frac{1}{2}(n^2 - 3n + 2)$ of the $\binom{n}{3}$ triangles need to be contained in a minimum cycle basis; they are all interchangeable.

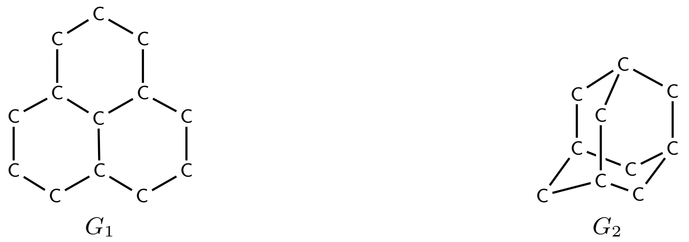


Fig. 3. Molecules $\boldsymbol{G}_1$ (Perhydrophenalene) $C_{13}H_{22}$ and $\boldsymbol{G}_2$ (Adamantane) $C_{13}H_{16}$. All edge weights are assumed to be equal to one. The minimum cycle basis of $\boldsymbol{G}_1$ consists of the three hexagons. Since they are unique, each hexagon represents its own $\sim_6$ equivalence class. In $\boldsymbol{G}_2$, any three of the four visible hexagons add up to the fourth, hence they are all interchangeable. Any three of them form a minimum cycle basis.

It is possible to compute the generally exponential interchangeability classes explicitly. However, the following observation leads to a graph invariant which can be computed in polynomial time:

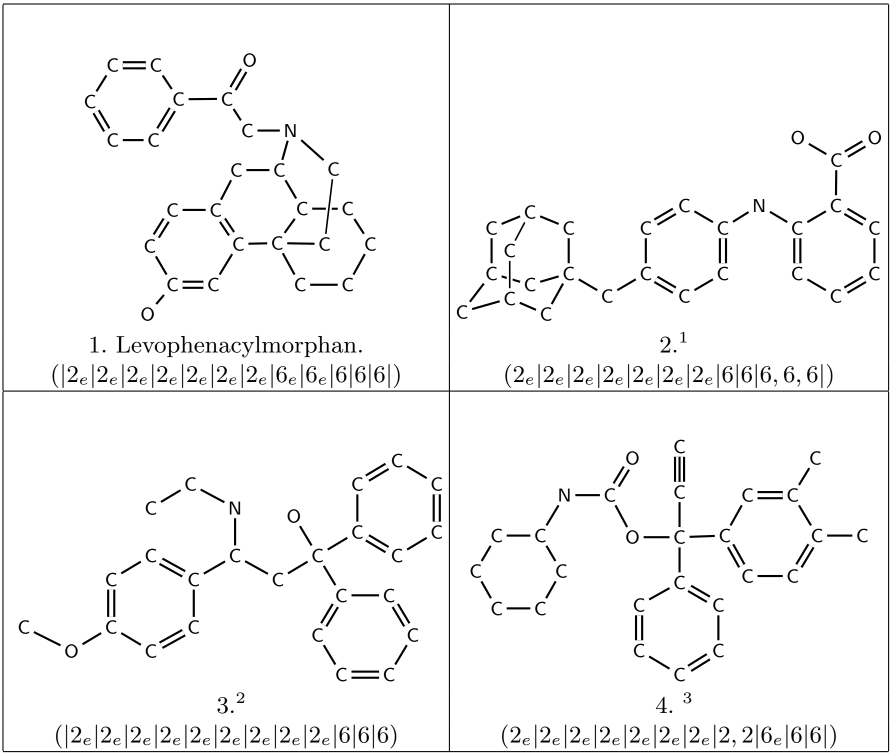
Lemma 8 ([33]). *Let $\kappa > 0$ be the weight of some relevant cycle, let $\mathcal{B}, \mathcal{B}'$ be two different minimum cycle bases and let $\mathcal{W}^\kappa$ be an equivalence class for $\sim_\kappa$. Then $|\mathcal{B} \cap \mathcal{W}^\kappa| = |\mathcal{B}' \cap \mathcal{W}^\kappa|$.*

Therefore we may call $|\mathcal{B} \cap \mathcal{W}^\kappa|$ the *relative rank* of $\mathcal{W}^\kappa$. This rank describes how many cycles of equal weight in a minimum cycle basis are related with respect to interchangeability. For each weight κ of a relevant cycle, there may be several equivalence classes. (Graph $\boldsymbol{G}_1$ in Fig. 3 has three $\sim_6$ equivalence classes, each with relative rank 1, Graph $\boldsymbol{G}_2$ has one $\sim_6$ equivalence class of relative rank 3.) The ordered vector $\vec{\beta}(\boldsymbol{G})$ containing the relative ranks of the $\sim_\kappa$ equivalence classes is a graph invariant. Thus, intuitively, $\vec{\beta}(\boldsymbol{G})$ measures the degree of interconnectivity or local density of $\boldsymbol{G}$'s cyclic structure (compare again the two graphs in Fig. 3).

Theorem 2 ([34]). *$\vec{\beta}(\boldsymbol{G})$ can be computed in time $O(m^4 n)$.*

We may encode the information gained from a minimum cycle basis and the relative ranks of the interchangeability classes within one vector: Vertical lines separate the entries in $\vec{w}(\boldsymbol{G})$ according to the $\sim_\kappa$ equivalence classes. The relative rank of each class corresponds to the number of entries between two vertical lines, sorted by increasing rank. A subscript e means that the corresponding equivalence class has cardinality 1.

In Figure 4, compare in particular graphs 1 and 2, and 3 and 4, respectively. For each pair, the vector $\vec{w}(\boldsymbol{G})$ is equal, but the difference can be noticed by looking at the information about the relative ranks.



1. Levophenacylmorphan.
$(|2_e|2_e|2_e|2_e|2_e|2_e|2_e|6_e|6_e|6|6|6|)$

2.[1]
$(2_e|2_e|2_e|2_e|2_e|2_e|2_e|6|6|6,6,6|)$

3.[2]
$(|2_e|2_e|2_e|2_e|2_e|2_e|2_e|2_e|2_e|6|6|6)$

4. [3]
$(2_e|2_e|2_e|2_e|2_e|2_e|2_e|2,2|6_e|6|6|)$

Fig. 4. Different molecules with the molecular formula $C_{24}H_{27}NO_2$ and their invariants

[1] N-(4-(1-adamantylmethyl)phenyl)anthranilic acid

[2] 1-,1-diphenyl-3-(ethylamino)-3-(p-methoxyphenyl)-1-propanol

[3] N-cyclohexyl-,1-phenyl-1-(3,4-xylyl)-2-propynyl ester

3.3 Periodic Event Scheduling

In periodic event scheduling problems, the events $v_1, \ldots, v_n$ have to be scheduled to occur periodically with period $T > 0$ subject to certain constraints [35,36,37,38,39].

A typical example is the construction of a bus or subway schedule in which a bus or subway services each station on its line at regular intervals of T minutes. A valid schedule will consist of instructions for each driver on each of the lines, at what time to arrive at and to leave from each station. There may be constraints on such a schedule, for instance, interconnection constraints between different lines. These could provide passengers with transfer options or prevent trains from arriving at the same time at a station for security reasons. Further, it is of course of interest to find a schedule which is also optimal with respect to a linear or nonlinear objective function, as for instance the average customer connection time.

A periodic event v consists of an infinite number of individual events $v = (v^i)_{i\in\mathbb{N}}$ such that

$$t(v^i) - t(v^{i-1}) = T$$

holds for the starting times $t(v^i)$ of the event v^i. Clearly, the starting time of a single individual event determines that of all other events in v. Thus, we define the *starting time* of v by

$$\pi(v) = \min\{t(v^i) + pT : i \in \mathbb{Z}, t(v^i) + pT \geq 0\} \in [0, T).$$

Spans allow the modeling of constraints: trivial constraints prevent, e.g., a train from leaving a station before it arrives; interconnection constraints may restrict the schedules of different lines: Given $l, u \in \mathbb{R}$ such that $0 \leq u - l < T$, a *span* is the union $[l, u]_T = \bigcup_{k\in\mathbb{Z}}[l + kT, u + kT]$. A span constraint $e = (v_i, v_j)$ is a relation between an ordered pair of periodic events and a span $[l_{ij}, u_{ij}]_T$ so that

$$\pi(v_j) - \pi(v_i) \in [l_{ij}, u_{ij}]_T.$$

The periodic event scheduling problem (PESP) is given by the tuple (T, V, E) where E is a set of span constraints. The task is to construct an (optimal) schedule $\pi : V \longrightarrow \mathbb{R}$ so that all span constraints in E are fulfilled. The problem is known to be NP-hard [35]. There exist various mixed integer programming (MIP) formulations. We describe one that is based on the event graph $G = (V, E)$, with vertex set $V = \{v_1, \ldots, v_n\}$ and oriented edges between v_i and v_j whenever $e = (v_i, v_j) \in E$. The edges are labeled with the spans $[l_{ij}, u_{ij}]_T$. A valid schedule can then be seen as a labeling of the vertices of the graph by $\pi(v_i)$ so that for each edge e and $p_{ij} \in \mathbb{Z}$,

$$\pi(v_j) + p_{ij} \cdot T \in \{\pi(v_i) + x_{ij} : x_{ij} \in [l_{ij}, u_{ij}]\}$$

for x_{ij} in the regular interval $[l_{ij}, u_{ij}]$. The label $\pi(v_i)$ is also called *potential* of v_i. With $x_{ij} = \pi(v_j) - \pi(v_i)$, the expression $x_{ij} + p_{ij} \cdot T$ is referred to as *periodic voltage*. Exploiting this analogy to electric networks and assuming that the underlying oriented graph is two-connected, a schedule is valid if and only if $\langle \boldsymbol{C}, x\rangle = 0$ for any cycle $\boldsymbol{C}$ in $\boldsymbol{G}$, where the vectors $x = (x_{ij}) \in \mathbb{R}^m, p = (p_{ij}) \in \mathbb{Z}^m$ fulfill the corresponding box constraints $l \leq x + pT \leq u$ with $l = l_{ij}$ and $u = (u_{ij})$. This gives rise to the MIP-formulation:

$$\begin{aligned} \min \quad & c(x + pT) \\ Ax &= 0 \\ l \leq x + pT &\leq u \\ p &\in \mathbb{Z}^m, \end{aligned}$$

where A denotes the cycle-edge incidence matrix of a $\mathbb{Q}$-cycle basis of $\boldsymbol{G}$ and $c(x + pT)$ is an objective function in terms of $x + pT$. The problem can be reformulated in more condensed form [40]; then A has to be the incidence matrix of an *integral* $\mathbb{Q}$-cycle basis, a cycle basis which uses only integer coefficients to generate all cycles in $\boldsymbol{G}$, see [41,20].

Experiments show that the problem can be solved faster when the chosen cycle basis is small. Several real-world case studies exist. For instance, the Berlin subway network and part of the Deutsche Bahn network was studied in [42,40], the Munich subway network was analyzed in [43].

4 Conclusion

Minimum and other cycle bases over different fields are interesting objects. The known algorithms all exploit the matroid property.

An open problem is to understand the exact difficulty frontier between minimum cycle bases and minimum cycle bases with an additional property: minimum cycle bases can be computed in polynomial time. Minimum fundamental tree cycle bases are hard to compute and to approximate, and minimum weakly fundamental cycle bases are hard to approximate, but it is not clear what happens in between: fundamental cycle bases are a proper subset of integral cycle bases, for instance, but currently, no polynomial time algorithm is known for computing a minimum integral cycle basis.

Acknowledgments. The authors were supported by the DFG Schwerpunktprogramm Nr. 1126, grants GR 933/8-1, GR 933/8-2. We thank the anonymous referees for some helpful remarks.

References

1. Biggs, N.: Algebraic Graph Theory. Cambridge University Press, Cambridge (1993)
2. Diestel, R.: Graph Theory. Graduate Texts in Mathematics, vol. 173. Springer, Heidelberg (1997)
3. Oxley, J.G.: Matroid Theory. Oxford University Press, Oxford (1992)
4. Paton, K.: An algorithm for finding a fundamental set of cycles of a graph. Comm. ACM 12, 514–519 (1969)
5. Hartvigsen, D., Zemel, E.: Is every cycle basis fundamental? J. Graph Theory 13, 117–137 (1989)
6. Deo, N., Prabhu, G., Krishnamoorthy, M.: Algorithms for generating fundamental cycles in a graph. ACM Trans. Math. Softw. 8, 26–42 (1982)
7. Berger, F., Gritzmann, P., de Vries, S.: Minimum cycle bases for network graphs. Algorithmica 40, 51–62 (2004)
8. Horton, J., Berger, F.: Minimum cycle bases of graphs over different fields. Electronic Notes in Discrete Mathematics 22, 501–505 (2005)
9. Liebchen, C., Rizzi, R.: A greedy approach to compute a minimum cycle basis of a directed graph. Information Processing Letters 94, 107–112 (2005)
10. Hubicka, E., Sysło, M.: Minimal bases of cycles of a graph. In: Fiedler, M. (ed.) Recent Advances in Graph Theory, Proc. 2nd Czechoslovak Conference on Graph Theory, Academia, pp. 283–293 (1975)
11. Plotkin, M.: Mathematical basis of ring-finding algorithms in CIDS. J. Chem. Documentation 11, 60–63 (1971)
12. Horton, J.: A polynomial-time algorithm to find the shortest cycle basis of a graph. SIAM J. Comput. 16, 358–366 (1987)

13. Vismara, P.: Union of all the minimum cycle bases of a graph. Electronic J. Comb. 4, R9 (1997)
14. de Pina, J.: Applications of shortest path methods. Ph.D thesis, University of Amsterdam, The Netherlands (2002)
15. Golynski, A., Horton, J.D.: A polynomial time algorithm to find the minimum cycle basis of a regular matroid. In: Penttonen, M., Schmidt, E.M. (eds.) SWAT 2002. LNCS, vol. 2368, p. 200. Springer, Heidelberg (2002)
16. Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: A faster algorithm for minimum cycle bases of graphs. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 846–857. Springer, Heidelberg (2004)
17. Mehlhorn, K., Michail, D.: Minimum cycle bases: Faster and simpler. ACM Trans. Algorithms (2009) (to appear)
18. Kavitha, T., Mehlhorn, K.: Algorithms to compute minimum cycle basis in directed graphs. Theory of Computing Systems 40, 485–505 (2007)
19. Hartvigsen, D., Mardon, R.: The prism-free planar graphs and their cycles bases. J. Graph Theory 15, 431–441 (1991)
20. Berger, F.: Minimum cycle bases in graphs. Ph.D thesis, TU München (2004)
21. Amaldi, E., Liberti, L., Maculan, N., Maffioli, F.: Efficient edge-swapping heuristics for finding minimum fundamental cycle bases. In: Ribeiro, C.C., Martins, S.L. (eds.) WEA 2004. LNCS, vol. 3059, pp. 14–29. Springer, Heidelberg (2004)
22. Hartvigsen, D., Mardon, R.: When do short cycles generate the cycle space? J. Comb. Theory, Ser. B 57, 88–99 (1993)
23. Kavitha, T., Mehlhorn, K., Michail, D.: New approximation algorithms for minimum cycle bases of graphs. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 512–523. Springer, Heidelberg (2007)
24. Hariharan, R., Kavitha, T., Mehlhorn, K.: Faster deterministic and randomized algorithms for minimum cycle basis in directed graphs. SIAM J. Comp. 38, 1430–1447 (2008)
25. Liebchen, C., Rizzi, R.: Classes of cycle bases. Discrete Appl. Math. 55, 337–355 (2007)
26. Galbiati, G., Amaldi, E.: On the approximability of the minimum fundamental cycle basis problem. In: Solis-Oba, R., Jansen, K. (eds.) WAOA 2003. LNCS, vol. 2909, pp. 151–164. Springer, Heidelberg (2004)
27. Rizzi, R.: Minimum weakly fundamental cycle bases are hard to find. Algorithmica (2009), doi:10.1007/s00453–007–9112–8
28. Elkin, M., Liebchen, C., Rizzi, R.: New length bounds for cycle bases. Information Processing Letters 104, 186–193 (2007)
29. Chen, W.K.: Active Network Analysis. Advanced series in electrical and computer engineering, vol. 2. World Scientific, Singapore (1991)
30. Hachtel, G., Brayton, R., Gustavson, F.: The sparse tableau approach to network analysis and design. IEEE Transactions of Circuit Theory CT-18, 111–113 (1971)
31. Estévez-Schwarz, D., Feldmann, U.: Diagnosis of erroneous circuit descriptions. Slides of a presentation, Paderborn (March 2003)
32. Downs, G.M., Gillet, V.J., Holliday, J.D., Lynch, M.F.: Review of ring perception algorithms for chemical graphs. J. Chem. Inf. Comput. Sci. 29, 172–187 (1989)
33. Gleiss, P., Leydold, J., Stadler, P.: Interchangeability of relevant cycles in graphs. Electronic J. Comb. 7, R16 (2000)
34. Berger, F., Gritzmann, P., de Vries, S.: Computing cyclic invariants for molecular graphs (2008) (manuscript)
35. Serafini, P., Ukovich, W.: A mathematical model for periodic scheduling problems. SIAM J. Discrete Math. 2, 550–581 (1989)

36. Odijk, M.: Railway timetable generation. Ph.D thesis, TU Delft, The Netherlands (1997)
37. Nachtigall, K.: Periodic network optimization with different arc frequencies. Discrete Appl. Math. 69, 1–17 (1996)
38. Nachtigall, K., Voget, S.: Minimizing waiting times in integrated fixed interval timetables by upgrading railway tracks. Europ. J. Oper. Res. 103, 610–627 (1997)
39. Liebchen, C., Möhring, R.: A case study in periodic timetabling. In: Proceedings of ATMOS 2002. Elec. Notes in Theor. Comput. Sci (ENTS), vol. 66.6 (2002)
40. Liebchen, C., Proksch, M., Wagner, F.: Performance of algorithms for periodic timetable optimization. In: Computer-aided Systems in Public Transport. Lect. Notes in Economics and Mathematical System, vol. 600, pp. 151–180 (2008)
41. Liebchen, C.: Finding short integral cycle bases for cyclic timetabling. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 715–726. Springer, Heidelberg (2003)
42. Liebchen, C., Möhring, R.: The modeling power of the periodic event scheduling problem: railway timetables – and beyond. In: Computer-aided Systems in Public Transport, pp. 117–150 (2008)
43. Kaminski, A.: Erzeugung und Optimierung zyklischer Zeitpläne. Diploma Thesis (in German), TU München (2004)

A Survey on Approximation Algorithms for Scheduling with Machine Unavailability

Florian Diedrich[1,⋆,⋆⋆,⋆⋆⋆], Klaus Jansen[1,⋆⋆,⋆⋆⋆],
Ulrich M. Schwarz[1,⋆⋆,⋆⋆⋆], and Denis Trystram[2,†]

[1] Institut für Informatik, Christian-Albrechts-Universität zu Kiel,
Olshausenstr. 40, 24098 Kiel, Germany
{fdi,kj,ums}@informatik.uni-kiel.de

[2] LIG – Grenoble University, 51 avenue Jean Kuntzmann,
38330 Montbonnot Saint-Martin, France
denis.trystram@imag.fr

Abstract. In this chapter we present recent contributions in the field of sequential job scheduling on *network machines* which work in parallel; these are subject to temporary unavailability. This unavailability can be either unforeseeable (online models) or known a priori (offline models). For the online models we are mainly interested in preemptive schedules for problem formulations where the machine unavailability is given by a probabilistic model; objectives of interest here are the sum of completion times and the makespan. Here, the non-preemptive case is essentially intractable. For the offline models we are interested in non-preemptive schedules where we consider the makespan objective; we present approximation algorithms which are complemented by suitable inapproximability results. Here, the preemptive model is polynomial-time solvable for large classes of settings.

1 Introduction

One major application that would not be possible without large communication networks is distributed computing where machines work in parallel: in large-scale projects such as SETI@home [4], the Internet Mersenne Prime Project [63] or medical applications [5,6], processing power is donated by volunteers who receive subjobs transmitted over the internet, calculate and return the results. In this perspective, individual units constitute a dynamic distributed computational

[⋆] Research supported in part by a grant "DAAD Doktorandenstipendium" of the German Academic Exchange Service. Part of this work was done while visiting the LIG, Grenoble University.

[⋆⋆] Supported in part by EU research project AEOLUS, "Algorithmic Principles for Building Efficient Overlay Computers", EU contract number 015964.

[⋆⋆⋆] Supported in part by DFG priority program 1126, "Algorithmics of Large and Complex Networks".

[†] Part of this work was supported by the "CoreGRID" Network of Excellence supported by the EU.

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 50–64, 2009.

environment which is often termed as *overlay computing platform* or simply *grid.* Clearly it is not appropriate to make constraining demands in terms of processing power from those participants. Furthermore, in a dynamic network, reachability of the participants cannot be guaranteed. This practical demand leads to a host of new problems in parallel machine scheduling.

Hence, due to the dynamic nature of the computing platform, a crucial issue is to take intervals of machine unavailability into account. This restriction in availability of computational resources is also realistic due to periods of machine maintenance or because jobs of highest priority are present. In either case we obtain models capturing realistic industrial settings and scheduling problems in new parallel computing platforms.

This chapter is organized as follows. In Sect. 2 we discuss approaches and results for online models; here we first discuss existing results before presenting our contributions. Likewise, in Sect. 3 we give a survey on the literature concerning offline models and briefly sketch our new algorithmic techniques put into effect. Finally we conclude in Sect. 4 with open research problems.

2 Online Models

In this section, we give an overview of results for various online models, i.e. not all machine failures are known in advance; the jobs are given offline, i.e. all information concerning the jobs os known a priori. A common tool to obtain results is to think of an *adversary* who dictates external events to force the algorithm to obtain results as bad as possible. We shall start with models and results that can be found in the literature and outline more recent results in Sect. 2.2. Note that we do not consider here settings in which the job durations are random but rather refer the reader to [12,45,51] for recent results and surveys on this topic.

2.1 Preliminaries and Known Results

In the following, we let n the number of jobs, and we denote with m the number of machines. In most online settings, we assume that m is known a priori. The processing times of the jobs are denoted $p_1, \ldots, p_n$. A schedule is an assignment of jobs to machines such that at no time, a job is executed on more than one machine and no machine executes more than one job. In a *preemptive* schedule, processing of a job can be interrupted and resumed at a later time, even on a different machine, while in a *non-preemptive* schedule, this is not permitted. Different objectives can be considered, the most common are the latest completion time C_j of any job j, called *makespan*, and the *average completion time* (or, equivalently, the sum of completion times $\sum C_j$). If for each job, a deadline or due date d_j is given by which it should be completed, we can also define the *tardiness* of a job as the amount of time by which it misses its deadline.

Negative results and restricted models. It is perhaps not very surprising that there are no positive results that concern non-preemptive scheduling with online failures: without further restriction, the adversary can prevent any algorithm even from scheduling jobs at all.

Example 1. Consider two identical machines M_1, M_2, two jobs with processing time $p_1 = 1, p_2 = 2$, and an arbitrary algorithm A. At time $t = 0$, both machines are available. Let t_2 be the time at which A starts the second job J_2, w.l.o.g. on machine M_1. The adversary now shuts off M_1 at time $t_2 + 1$ and M_2 at time $t_2 + 2$. It is obvious that J_2 cannot terminate under A, and an offline optimal algorithm could schedule J_1 on M_1 and J_2 on M_2 at time t_2.

In total, this means that for both objectives $C_{\max}$ and $\sum C_j$ under consideration, no bounded competitive ratio can be achieved. Even if we allow preemptions, there are two major settings to distinguish here: in the "classical" preemptive setting, jobs can continue from the point they were interrupted, without any loss or penalty due to interruption. A stronger, but more realistic, setting requires that an interrupted job is *restarted* from the beginning. It is easy to see that many examples for the non-preemptive setting will carry over to the model with restarts. Therefore, for online problems, we concentrate on the truly preemptive setting.

As the following example from [3] shows, shutting off all machines is very powerful, even when the algorithm is permitted preemptions:

Example 2. Consider an instance with m identical machines $M_1, \ldots, M_m$, and m jobs $J_1, \ldots, J_m$ of unit processing time. Initially, only M_1 is available. Let t the first time when some job, w.l.o.g. J_1, is started by some algorithm A, and t' the time when J_1 first ceases to run, either by termination or by preemption. At time t', machines $M_2, \ldots, M_m$ become available, and all machines will become unavailable at time $t' + 1 - (t' - t)/m$. Clearly, A cannot complete all jobs by time $t' + 1 - (t' - t)/m$. However, an offline optimal algorithm could share the interval $[t, t')$ on machine M_1 between all jobs, thus being able to complete all jobs.

Note that in this example, the algorithms will terminate if we force an unlimited amount of processing time to be available, but their competetive ratio will be arbitrarily bad if there are large intervals of time where no machine is available. Hence, most settings will require that at all times, at least one machine is available. This guarantees that even simple one-machine algorithms are $\mathcal{O}(m)$-competitive.

A different restriction limits the adversary's power to change machine availability suddenly: such changes must be announced beforehand, either in the way that the time of the next change of availability is known (lookahead) or that all changes for a fixed window into the future are known (rolling horizon). These two settings are in many cases closely related: consider a look-ahead algorithm in a rolling horizon setting. Clearly, the algorithm can know the time of the next event if it is within the horizon. To make sure that there is always an event within the current planning horizon, we insert dummy events, for example, a machine leaves and immediately recovers, at regular intervals. (This is not a restriction in settings that do not penalize preemption.)

On the other hand, consider a rolling-horizon algorithm in a look-ahead setting. Depending on the circumstances, it is often possible to change the length of the horizon on the fly, as long as its length is lower-bounded to guarantee termination. Now, we can present the time interval from the present time to the next event as a horizon to the algorithm and run the algorithm. After this, we are told the next event and again present a fixed horizon, of possibly different length, to the algorithm and so on.

One final restriction that can be made concerns the selection of machines that can be disabled or enabled. Of particular interest in the online setting are three patterns:

1. In a *zig-zag* pattern, the number of machines at time t, $m(t)$, is always m or $m-1$ for some suitable constant $m \in \mathbb{N}$.
2. In an *increasing* pattern, for all two points in time t, t' with $t < t'$, we have $m(t) \leq m(t')$, i.e. the number of available machines never decreases.
3. In an *increasing zig-zag* pattern, for $t < t'$, we have $m(t) - 1 \leq m(t')$. Intuitively, if we have $m(t)$ machines now, we will always have at least $m(t) - 1$ machines in the future.

Known positive results. Not many results are known for the online setting, mostly considering the makespan and some only for the single-machine case. Kasap et al. [33] have shown that for the single-machine problem, the "Longest First" heuristic is optimal in expectation if the uptime distribution (i.e. the time between failures) is convex. Adiri et al. [1] have given asymptotically optimal algorithms to minimize the expected sum of completion times if the breakdowns are exponentially distributed. These settings are not preemptive in our sense, since an interrupted job must be started anew. (In the single-machine case, results with resumable jobs are generally not harder than their corresponding counterparts without unavailability constraints.) Li & Cao [46] have considered a setting with two different kinds of breakdowns: one kind which allows resumption, and one kind which forces restart, possibly with entirely different job characteristics. They give results for weighted completion time if all random variables are exponentially distributed.

Less is known about the multi-machine case: Lee & Yu [44] consider the case in which all machines will fail at the same time, but the length of the disruption is unknown. They present pseudopolynomial algorithms to minimize the weighted sum of completion times for both the resumable and restart settings. In the case that there is a lookahead to the next event, Albers & Schmidt [3] have shown that the optimal makespan can be achieved by a simple "Longest Remaining First" heuristic. In [60], this result is extended to related machines and non-zero release times by using the corresponding extension "Longest Remaining First on Fastest Machine". One can remove the need for lookahead and still obtain a makespan of OPT $+ \epsilon$ for any fixed $\epsilon > 0$ for identical machines. This, too, can be extended to related machines [59], however, the number of preemptions will then depend polynomially on the ratio of maximal to minimal machine speed.

Earlier results of Sanlaville [57] are concerned with the maximum tardiness objective. (This is a more general case than the makespan, since we may add due dates of 0 for all jobs.) If additionally all the release times are 0, then the *shortest laxity first* heuristic yields optimal results for zig-zag availability. In this heuristic, the priority of a job is higher if its remaining processing time is "high" and its deadline is "soon" (i.e. we consider the values $d_i - \text{rem}_i(t)$ for a time t and select the jobs of smallest difference). If the release times are not all equal, this heuristic still yields a schedule of value at most $\text{OPT} + \max p_j$. Some results are known if there are precedence constraints on the jobs: if the constraints form chains, the optimal makespan is found by a "Longest Path First" heuristic, as shown by Liu & Sanlaville [48]. This algorithm is still optimal for outforests (each job has at most one immediate predecessor) and decreasing zig-zag patterns (and vice versa for inforests and increasing zig-zag). Note that such heuristics are suitable for online algorithms since all jobs are known from the beginning, and only the machine availability is online.

These results only hold for the case that preemption and migration are permitted, i.e. we may interrupt execution of jobs at any time and resume it – even on another machine – with no penalty. In particular, this means that a machine failure will only negatively affect the future. Kalyanasundaram & Pruhs [32] have considered the restart model: whenever a machine fails, the progress it has made on the job it was executing is discarded completely. To overcome this problem, the same job is executed several times, possibly in parallel, in hope that one of the copies will eventually succeed. In particular, they show:

Theorem 1. *If each machine fails independently with probability f and machines do not recover, the competitive ratio of a deterministic online algorithm for the $C_{\max}$ objective is $\Omega(\log m/(\log\log m))$; for the $(1/n)\sum C_j$ objective, it is $\Omega(1)$. Both these bounds are tight in the sense that there exist algorithms of competitive ratios $\mathcal{O}(\log m/(\log\log m))$ resp. $\mathcal{O}(1)$.*

Note that here the failure probability is a parameter dependent on the time step; the modelization of uptime distribution from data of actual systems is an interesting topic of its own [53].

2.2 New Results

Some results were recently found by the authors in [62], mostly concerning the sum of completion times for identical machines. The main techniques used are rearrangment of preempted portions of jobs and a suitable reduction to a one-machine schedule by means of regarding the schedule as a queue. In particular, they show the following result:

Theorem 2. *The heuristic SRPT, which at any time executes those jobs which have the shortest remaining processing time, minimizes $\sum C_j$ for increasing zig-zag availability pattern.*

Corollary 1. *SRPT minimizes $\sum C_j$ on two machines, if one of them is always available.*

However, SRPT is not a good heuristic for arbitrary patterns:

Example 3. Consider an instance with m machines (where m is even) and $m + m/2$ jobs. $J_1, \ldots, J_m$ have length 1, $J_{m+1}, \ldots, J_{m+m/2}$ have length 2. During the interval $[0, 2)$, all machines $M_1, \ldots, M_m$ are available, and after that, only M_1 is available.

SRPT will schedule all short jobs immediately and hence will not complete all jobs by time 2. The optimal solution is to start all long jobs immediately and fill empty machines with short jobs. It is easily verified that the ratio of the sums of completion times is then $\Omega(m)$.

Indeed, we can show [61]:

Theorem 3. No *online algorithm can be* $(2 - \epsilon)$*-competitive for* $\sum C_j$*, for any* $\epsilon > 0$*, even if all job lengths are in* $\{1, 2\}$.

Examination of the bad instances we have given here reveals that they depend heavily on the ability to shut off many (usually: all but one) machines at the same time. This leads to a different performance measure of the algorithms where the probability of machine failure influences the quality.

If we consider the setting where each of the m machines fails independently of the others with some probability f that is independent of the machine, the following result can be shown [62]:

Theorem 4. *Given a scheduling problem on identical machines which fail with probability* f*, and an* α*-approximate algorithm for the offline setting without failures, there is an online algorithm with asymptotic competitive ratio* $\alpha/(1-f)$*.*

This holds for both makespan and weighted sum of completion times, even if the instance also has release times and precedence constraints.

Table 1 lists some suitable offline algorithms and the setting they pertain to.

Table 1. Some offline results that can be adapted to online settings. First column describes scheduling problems in 3-field notation, cf. [45,54].

Setting	Source		Approximation ratio
$\mathrm{P}\vert pmtn\vert \sum C_j$	McNaughton	[50]	1
$\mathrm{P}\vert\vert \sum w_j C_j$	Kawaguchi & Kyan	[34]	$(1 + \sqrt{2})/2$
$\mathrm{P}\vert r_j, pmtn\vert \sum w_j C_j$	Afrati et al.	[2]	PTAS
$\mathrm{P}\vert r_j, prec, pmtn\vert \sum w_j C_j$	Hall et al.	[22]	3

3 Offline Models

Here we discuss recent offline models where the job characteristics as well as the periods of machine unavailability are given in advance. We give a brief survey on approximation algorithms complemented by inapproximability results; these can be summarized as in Tab. 2; we exclusively study the makespan objective for our problems here.

Table 2. Complexity results for offline problems in 3-field notation, cf. [45,54]

Problem	$m = 1$	$m = 2$	$m \geq 3$
$\mathrm{P}m \mid nr\text{-}a \mid C_{\max}$ arbitrary reservations	no polynomial time algorithm with constant approximation ratio unless $\mathsf{P} = \mathsf{NP}$		
$\mathrm{P}m \mid nr\text{-}a \mid C_{\max}$ at most one reservation per machine	NP-hard, FPTAS	no polynomial time algorithm with constant approximation ratio unless $\mathsf{P} = \mathsf{NP}$	
$\mathrm{P}m, 1up \mid nr\text{-}a \mid C_{\max}$ arbitrary reservations	P $(r = 0)$	strongly NP-hard, PTAS, no FPTAS unless $\mathsf{P} = \mathsf{NP}$	
$\mathrm{P}m, 1up \mid nr\text{-}a \mid C_{\max}$ at most one reservation per machine	P $(r = 0)$	NP-hard, FPTAS	strongly NP-hard, PTAS, no FPTAS unless $\mathsf{P} = \mathsf{NP}$
$\mathrm{P}, 1up \mid nr\text{-}a \mid C_{\max}$ at most one reservation per machine	no polynomial time algorithm with approximation ratio better than $3/2$ unless $\mathsf{P} = \mathsf{NP}$		
$\mathrm{P}, 1up \mid nr\text{-}a \mid C_{\max}$ where a constant percentage of machines is assumed to be permanently available	no polynomial time algorithm with approximation ratio better than $3/2$ unless $\mathsf{P} = \mathsf{NP}$, approximation algorithm with a PTAS-like running time and approximation ratio $3/2 + \epsilon$ for any $\epsilon \in (0, 1/2]$		

This section is organized as follows. In Subsect. 3.1, we discuss preliminaries and related work. In Subsect. 3.2, we present some inapproximability results which are complemented in Subsect. 3.3 by suitable approximation algorithms. Techniques used to obtain our new results [15,17] are mainly dual approximation via binary search over a suitable target makespan, a PTAS for Multiple Subset Sum, definition of configurations, linear grouping and rounding known from state-of-the-art approximation schemes for Bin Packing, and an interesting cyclic shifting argument which permits usage of a network flow model for assignment of large jobs.

3.1 Preliminaries and Known Results

Our problem can be formally defined as follows. Let $m \in \mathbb{N}^*$ denote the number of machines which is assumed to be either constant or part of the input; this results in different formulations, however. Besides the n jobs, an instance I consists of r reservations $R_1, \ldots, R_r$. For each $k \in \{1, \ldots, r\}$, $R_k = (i_k, s_k, t_k)$ indicates unavailability of machine M_{i_k} in the time interval $[s_k, t_k)$, where we have $s_k, t_k \in \mathbb{N}, i_k \in \{1, \ldots, m\}$ and $s_k < t_k$. We suppose that for reservations on the same machine there is no overlap; for two reservations $R_k, R_{k'}$ such that $i_k = i_{k'}$ holds, we have $[s_k, t_k) \cap [s_{k'}, t_{k'}) = \emptyset$. For each $i \in \{1, \ldots, m\}$ let $R_i' := \{R_k \in I | i_k = i\}$

denote the set of reservations for machine M_i. Our goal is to compute a non-preemptive schedule of the jobs such that no job is scheduled on a machine that is unavailable and on each machine at most one job runs at a given time; the objective is to minimize the makespan. (We note that in contrast to the online setting, preemptions allow very large classes of problems to be solved exactly in polynomial time with LP and network flow methods as in [11,14].)

Using the 3-field notation, we denote the corresponding model by $\mathrm{P}m|nr\text{-}a|C_{\max}$ if m is a constant and by $\mathrm{P}|nr\text{-}a|C_{\max}$ if m is considered part of the input. One might argue that denoting the presence of unavailability intervals in the job field of the classification scheme might be counter-intuitive; however, reservations can be viewed as jobs owned by other users and this notation is well-established in the existing literature [41,42,43,45].

Concerning previous results, Lee [41] and Lee et al. [43] studied identical parallel machines which may have different starting times; here, the LPT policy, where jobs are started in non-increasing order of their length, was analyzed. Lee [42] studied the case where at most one reservation per machine is permitted while one machine is continuously available and obtained suitable approximation ratios for low-complexity list scheduling algorithms. Liao et al. [47] presented an experimental study of an exact algorithm for $m = 2$ within the same scenario. Hwang et al. [26] studied the LPT policy for the case where at most one interval of unavailability per machine is permitted. They proved a tight bound of $1 + \lceil m/(m - \lambda)\rceil/2$ where at most $\lambda \in [m - 1]$ machines are permitted to be unavailable simultaneously. The reader can find in [45], Chapt. 22, problem definitions and a more comprehensive survey about previous results; there, mostly NP-completeness proofs, fast greedy algorithms and dynamic programming formulations for various objectives can be found. In [58], Scharbrodt et al. present approximation schemes and inapproximability results for a setting where the reservations are seen as fixed jobs which consequently also contribute to the makespan. Note that the objective is quite different from the objective discussed in this section; the objective in [58] models makespan minimization from the perspective of the *system administrator* while the objective presented here models makespan minimization from the viewpoint of an *individual user* submitting his jobs to a system subject to preallocation of jobs.

For the model of interest, the sum of completion time objective has also been considered. In [52], a setting where at most one reservation per machine is permitted was studied; Mellouli et al. experimentally evaluated several exact algorithms. Furthermore, Kacem et al. [31] studied the corresponding single machine problem, also evaluating several exact algorithms; he also obtained an approximation algorithm with ratio 2 and running time $\mathcal{O}(n^2)$ for the same problem in [30]; a similar model was studied by Sadfi et al. in [55].

Until recently, makespan minimization for the model under consideration has not been approached with approximation schemes, not even for the special cases which have already been studied [26,42,47]. However, Kacem [29] studied the single machine problem with one reservation and tails; this is a model where jobs occupy the machine for a certain time, the tail, after their completion.

He obtained a 3/2-approximation algorithm and an FPTAS based on dynamic programming.

3.2 Inapproximability Results

In this section we present hardness and inapproximability results for the models under consideration. First it is clear that $\mathrm{P}m|nr\text{-}a|C_{\max}$ as well as $\mathrm{P}|nr\text{-}a|C_{\max}$ are NP-hard since they are generalizations of the corresponding problems $\mathrm{P}m||C_{\max}$ and $\mathrm{P}||C_{\max}$ without reservations. In addition, the problems under consideration are also hard to approximate, as shown in [15,16,17]; the respective constructions for the proofs use reductions from NP-complete partition problems [21] in combination with gap creation arguments.

Theorem 5. *Both problems* $\mathrm{P}m|nr\text{-}a|C_{\max}$ *and* $\mathrm{P}|nr\text{-}a|C_{\max}$ *do not admit a polynomial time algorithm with a constant approximation ratio unless* $\mathsf{P} = \mathsf{NP}$ *holds.*

Similar to the parallel setting studied in [18], for both of these problems the inapproximability is due to the existence of intervals in which no machine is available. Consequently no work can be carried out in these intervals; since the setting is non-preemptive, this causes a large undesired delay for some jobs. In total, it is necessary to find suitable restrictions which permit constant approximation ratios. One straightforward way to do this is to require at least one machine to be permanently available. The resulting problems are denoted by $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ and $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$, where $1up$ means that at least one machine is always available. These restricted problem formulations then are polynomially solvable for $m = 1$ and remain NP-hard for $m \geq 2$, which can be seen by defining a reservation for each machine except the first one and following the lines of the proof of Lemma 1 in [17]. As we discuss in Subsect. 3.3, $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ admits a PTAS [17]. On the other hand $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ does not admit an FPTAS unless $\mathsf{P} = \mathsf{NP}$; this result follows from the fact that $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ is strongly NP-hard [17] via a reduction from 3-Partition and the subsequent standard arguments for "well-behaved" objective functions as established in [20].

Theorem 6. *The problem* $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ *does not admit an FPTAS for* $m \geq 2$ *unless* $\mathsf{P} = \mathsf{NP}$ *holds.*

Furthermore it is a natural question whether $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ becomes easier if the number of reservations per machine is restricted to one. Surprisingly, this is not the case, which can be shown by adaptation of a construction from [7]. The following result implies that $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ with at most one reservation per machine for $m \geq 3$ is strongly NP-hard as well.

Theorem 7. *The problem* $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ *does not admit an FPTAS, even if there is at most one reservation per machine, for* $m \geq 3$ *unless* $\mathsf{P} = \mathsf{NP}$ *holds.*

However, $\mathrm{P}2, 1up|nr\text{-}a|C_{\max}$, if there is at most one reservation per machine, is easier to approximate in the sense that it admits an FPTAS, as we discuss in

Subsect. 3.3. Finally, the one-machine problem $1|nr\text{-}a|C_{\max}$ does not admit a polynomial time approximation algorithm if more than one reservation is permitted, but admits an FPTAS for only one reservation [17].

For the problem $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$ we obtain a much stronger inapproximability result than the strong NP-hardness of $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$. Based on a construction from [58] it can be shown that for any $\epsilon \in (0, 1/2]$ the problem $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$ does not admit a polynomial-time approximation algorithm with approximation ratio $3/2 - \epsilon$ unless $\mathsf{P} = \mathsf{NP}$ holds; the proof is based on a reduction from Numerical Matching with Target Sums [21]. Surprisingly, this lower bound is also valid if a constant percentage of the machines is assumed to be permanently available [15], which complements the sophisticated algorithm mentioned in Subsect. 3.3.

Theorem 8. *The problem* $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$ *does not admit an approximation algorithm with ratio* $3/2 - \epsilon$*, for any* $\epsilon \in (0, 1/2]$*, not even if the percentage of permanently available machines is constant, unless* $\mathsf{P} = \mathsf{NP}$ *holds.*

From a broader perspective, the hardness is due to the approximation of $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ with a constant ratio, even if there is at most one reservation per machine, being at least as hard as approximation of Bin Packing with an additive error; however whether the latter is possible is an open problem, discussed in [23], Chapt. 2, page 67.

Theorem 9. *If there is a polynomial time algorithm for* $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$ *with approximation ratio* $c \in \mathbb{N} \setminus \{1\}$*, then there is a polynomial time algorithm for Bin Packing with additive error* $2(c - 1)$*.*

3.3 Approximation Algorithms

To obtain approximation algorithms for the problems under consideration, different techniques from algorithm design are put into effect. The most basic approach used in [17] is based on dual approximation [24] and multiple subset sum problems. The latter ones are special cases of knapsack problems, which belong to the oldest problems in combinatorial optimization and theoretical computer science. Hence, we benefit from the fact that they are relatively well understood. For the classical problem (KP) with one knapsack, besides the result by Ibarra & Kim [27], Lawler presented a sophisticated FPTAS [40] which was later improved by Kellerer & Pferschy [37]; see also the textbooks by Martello & Toth [49] and Kellerer et al. [38] for surveys. The case where the item profits equal their weights is called the subset sum problem and denoted as SSP. The problem with *multiple* knapsacks (MKP) is a natural generalization of KP; the case with multiple knapsacks where the item profits equal their weights is called the *multiple* subset sum problem (MSSP). Various special cases and extensions of these problems have been studied [7,8,9,10,13,28,35,36], finally yielding PTASes and FPTASes for the cases upon which our approach is partially based [8,10,28,36].

As discussed in detail in [17], we obtain a PTAS for $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$ by using binary search over the target makespan and using a PTAS for MSSP where the capacities of the knapsacks are permitted to be different [8]; this complements the inapproximability result in Subsect. 3.2 by which the PTAS is, in a certain sense, a best possible algorithm. Furthermore, instead of a PTAS for MSSP we can use a fast greedy algorithm. This results in an algorithm for $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ with an approximation ratio of $1 + m/2$. In [26], the authors studied the case where at most one reservation per machine is permitted while λ machines are permanently available. They proved that for this setting LPT yields a tight bound of $1 + \lceil 1/(1 - \lambda/m)\rceil/2$. For $\lambda = m - 1$, this also yields $1 + m/2$. In total, we obtain the same approximation ratio for a much more general problem formulation, which at comes the cost of a larger runtime bound however.

Theorem 10. *The problem* $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ *admits a PTAS and a fast greedy algorithm with approximation ratio* $1 + m/2$.

If we study $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ but permit at most one reservation per machine, the situation is different; this change of complexity is illustrated in Tab. 2. The algorithm for $\mathrm{P}2, 1up|nr\text{-}a|C_{\max}$ mentioned in the next theorem is based on a dynamic programming formulation which combinatorializes the loads of the two machines in suitable intervals; in combination with a 2-approximation algorithm and scaling and rounding the values similar to [40] this dynamic programming formulation can be transformed into an FPTAS with a standard approach. Again this approximation scheme is to be compared with the complementing hardness result from Subsect. 3.2.

Theorem 11. *The problem* $\mathrm{P}2, 1up|nr\text{-}a|C_{\max}$ *with one reservation admits an FPTAS.*

For $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$ we have obtained an inapproximability result in Subsect. 3.2. However, in [15], we have obtained an approximation algorithm with ratio $3/2 + \epsilon$ for the case where the percentage of available machines is constant.

Theorem 12. *The problem* $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$, *provided that the percentage of permanently available machines is constant, admits a polynomial time algorithm with approximation ratio* $3/2 + \epsilon$ *for any* $\epsilon \in (0, 1/2]$.

The running time of the respective algorithm is PTAS-like, i.e. exponential in $1/\epsilon$; however, various interesting algorithmic techniques are used. These include dual approximation by guessing the makespan [24], linear grouping and rounding known from Bin Packing [19], enumeration of configurations, grouping and embedding known from Strip Packing [39] in combination with a novel method of assignment of jobs via network flow models and again a PTAS for MSSP as in [17].

4 Conclusion

We have given a survey on recent advances in scheduling with machine unavailability constraints; this is an intriguing field where online formulations are as relevant as classical offline formulations. One interesting problem in the online setting is the rather large gap for the makespan problem: while only 2-inapproximability is known, we do not know of any algorithm with constant competitiveness. Permission of reservations in the classical offline problems changes the complexity in a surprising way. More precisely, $\mathrm{P}m||C_{\max}$ is NP-hard but permits an FPTAS [56], while $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$ is strongly NP-hard for $m \geq 2$; furthermore, $\mathrm{P}||C_{\max}$ is strongly NP-hard but permits a PTAS [25], while $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$ is much harder to approximate. Concerning offline problems, it would be interesting to perform a sensitivity analysis with respect to the reservations for which a slight shift in time or increase and decrease in duration might be permitted. In total, we like to point out that here especially the algorithms for online problems tend to be suitable for implementation. This is to be compared with those for the offline problems which are rather suitable for long-term planning, where calculation of the schedule is not a time-critical issue.

Acknowledgements. The authors thank Érik Saule and Derrick Kondo for fruitful discussions and the referees for many helpful comments.

References

1. Adiri, I., Bruno, J.L., Frostig, E., Kan, A.H.G.R.: Single machine flow-time scheduling with a single breakdown. Acta Inf. 26(7), 679–696 (1989)
2. Afrati, F.N., Bampis, E., Chekuri, C., Karger, D.R., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., Sviridenko, M.: Approximation schemes for minimizing average weighted completion time with release dates. In: FOCS, pp. 32–44 (1999)
3. Albers, S., Schmidt, G.: Scheduling with unexpected machine breakdowns. Disc. App. Math. 110(2-3), 85–99 (2001)
4. Anderson, D.P., et al.: Seti@home, `http://setiathome.berkeley.edu/`
5. Baker, D., et al.: Rosetta@home protein folding, design and docking, `http://boinc.bakerlab.org/rosetta/`
6. Berman, F., et al.: World community grid, `http://www.worldcommunitygrid.org/`
7. Caprara, A., Kellerer, H., Pferschy, U.: The multiple subset sum problem. Technical report, Technische Universität Graz (1998)
8. Caprara, A., Kellerer, H., Pferschy, U.: A PTAS for the multiple subset sum problem with different knapsack capacities. Inf. Process. Lett. 73(3-4), 111–118 (2000)
9. Caprara, A., Kellerer, H., Pferschy, U.: A 3/4-approximation algorithm for multiple subset sum. J. Heuristics 9(2), 99–111 (2003)
10. Chekuri, C., Khanna, S.: A polynomial time approximation scheme for the multiple knapsack problem. SIAM J. Comput. 35(3), 713–728 (2005)
11. Cochand, M., de Werra, D., Slowinski, R.: Preemptive scheduling with staircase and piecewise linear resource availability. Methods and Models of Op. Res. 33, 297–313 (1989)
12. Crutchfield, C.Y., Dzunic, Z., Fineman, J.T., Karger, D.R., Scott, J.H.: Improved approximations for multiprecessor scheduling under uncertainty. In: Proceedings of SPAA (2008) (to appear)
13. Dawande, M., Kalagnanam, J., Keskinocak, P., Salman, F.S., Ravi, R.: Approximation algorithms for the multiple knapsack problem with assignment restrictions. J. Comb. Optim. 4(2), 171–186 (2000)

14. de Werra, D.: On the two-phase method for preemptive scheduling. Eur. J. Operational Res. 37, 227–235 (1988)
15. Diedrich, F., Jansen, K.: Improved approximation algorithms for scheduling with fixed jobs. In: Proc. 20th ACM-SIAM Symposium on Discrete Algorithms (2009) (to appear)
16. Diedrich, F., Jansen, K., Pascual, F., Trystram, D.: Approximation algorithms for scheduling with reservations. (unpublished Manuscript)
17. Diedrich, F., Jansen, K., Pascual, F., Trystram, D.: Approximation algorithms for scheduling with reservations. In: Aluru, S., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2007. LNCS, vol. 4873, pp. 297–307. Springer, Heidelberg (2007)
18. Eyraud-Dubois, L., Mounié, G., Trystram, D.: Analysis of scheduling algorithms with reservations. In: IPDPS, pp. 1–8. IEEE, Los Alamitos (2007)
19. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1 + \epsilon$ in linear time. Combinatorica 1(4), 349–355 (1981)
20. Garey, M.R., Johnson, D.S.: "strong" NP-completeness results: Motivation, examples, and implications. J. ACM 25(3), 499–508 (1978)
21. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
22. Hall, L.A., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: Off-line and on-line algorithms. In: SODA, pp. 142–151 (1996)
23. Hochbaum, D. (ed.): Approximation Algorithms for NP-hard Problems. PWS Publishing Company (1996)
24. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. J. ACM 34(1), 144–162 (1987)
25. Hochbaum, D.S., Shmoys, D.B.: A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. SIAM J. Comput. 17(3), 539–551 (1988)
26. Hwang, H.-C., Lee, K., Chang, S.Y.: The effect of machine availability on the worst-case performance of LPT. Disc. App. Math. 148(1), 49–61 (2005)
27. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. J. ACM 22(4), 463–468 (1975)
28. Jansen, K.: Parameterized approximation scheme for the multiple knapsack problem. In: Proc. 20th ACM-SIAM Symposium on Discrete Algorithms (2009) (to appear)
29. Kacem, I.: Approximation algorithms for the makespan minimization with positive tails on a single machine with a fixed non-availability interval. J. Comb. Optim. (2007)
30. Kacem, I.: Approximation algorithm for the weighted flow-time minimization on a single machine with a fixed non-availability interval. Computers & Industrial Engineering 54(3), 401–410 (2008)
31. Kacem, I., Chu, C., Souissi, A.: Single-machine scheduling with an availability constraint to minimize the weighted sum of the completion times. Computers & OR 35(3), 827–844 (2008)
32. Kalyanasundaram, B., Pruhs, K.: Fault-tolerant scheduling. SIAM Journal on Computation 34(3), 697–719 (2005)
33. Kasap, N., Aytug, H., Paul, A.: Minimizing makespan on a single machine subject to random breakdowns. Oper. Res. Lett. 34(1), 29–36 (2006)
34. Kawaguchi, T., Kyan, S.: Worst case bound of an LRF schedule for the mean weighted flow-time problem. SIAM Journal on Computation 15(4), 1119–1129 (1986)

35. Kellerer, H.: A polynomial time approximation scheme for the multiple knapsack problem. In: Hochbaum, D.S., Jansen, K., Rolim, J.D.P., Sinclair, A. (eds.) RANDOM 1999 and APPROX 1999. LNCS, vol. 1671, pp. 51–62. Springer, Heidelberg (1999)
36. Kellerer, H., Mansini, R., Pferschy, U., Speranza, M.G.: An efficient fully polynomial approximation scheme for the subset-sum problem. J. Comput. Syst. Sci. 66(2), 349–370 (2003)
37. Kellerer, H., Pferschy, U.: A new fully polynomial time approximation scheme for the knapsack problem. J. Comb. Optim. 3(1), 59–71 (1999)
38. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Heidelberg (2004)
39. Kenyon, C., Rémila, E.: A near-optimal solution to a two dimensional cutting stock problem. Math. Oper. Res. 25, 645–656 (2000)
40. Lawler, E.L.: Fast approximation algorithms for knapsack problems. Math. Oper. Res. 4(4), 339–356 (1979)
41. Lee, C.-Y.: Parallel machines scheduling with non-simultaneous machine available time. Disc. App. Math. 30, 53–61 (1991)
42. Lee, C.-Y.: Machine scheduling with an availability constraint. J. Global Optimization, Special Issue on Optimization of Scheduling Applications 9, 363–384 (1996)
43. Lee, C.-Y., He, Y., Tang, G.: A note on parallel machine scheduling with non-simultaneous machine available time. Disc. App. Math. 100(1-2), 133–135 (2000)
44. Lee, C.-Y., Yu, G.: Parallel-machine scheduling under potential disruption. Opt. Lett. 2(1), 27–37 (2008)
45. Leung, J.Y.-T. (ed.): Handbook of Scheduling. Chapman & Hall, Boca Raton (2004)
46. Li, W., Cao, J.: Stochastic scheduling on a single machine subject to multiple breakdowns according to different probabilities. Oper. Res. Lett. 18(2), 81–91 (1995)
47. Liao, C.-J., Shyur, D.-L., Lin, C.-H.: Makespan minimization for two parallel machines with an availability constraint. Eur. J. Operational Res. 160, 445–456 (2003)
48. Liu, Z., Sanlaville, E.: Preemptive scheduling with variable profile, precedence constraints and due dates. Disc. App. Math. 58(3), 253–280 (1995)
49. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. Wiley, Chichester (1990)
50. McNaughton, R.: Scheudling with deadlines and loss functions. Mgt. Science 6, 1–12 (1959)
51. Megow, N., Vredeveld, T.: Approximation in preemptive stochastic online scheduling. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 516–527. Springer, Heidelberg (2006)
52. Mellouli, R., Sadfi, C., Chu, C., Kacem, I.: Identical parallel-machine scheduling under availability constraints to minimize the sum of completion times. Eur. J. Operational Res. (2008)
53. Nurmi, D., Brevik, J., Wolski, R.: Modeling machine availability in enterprise and wide-area distributed computing environments. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 432–441. Springer, Heidelberg (2005)
54. Pinedo, M.: Scheduling: Theory, Algorithms and Systems. Prentice Hall, Englewood Cliffs (1995)
55. Sadfi, C., Penz, B., Rapine, C., Błażewicz, J., Formanowicz, P.: An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints. Eur. J. Operational Res. 161(1), 3–10 (2005)
56. Sahni, S.: Algorithms for scheduling independent tasks. J. ACM 23(1), 116–127 (1976)

57. Sanlaville, E.: Nearly on line scheduling of preemptive independent tasks. Disc. App. Math. 57(2-3), 229–241 (1995)
58. Scharbrodt, M., Steger, A., Weisser, H.: Approximability of scheduling with fixed jobs. J. Scheduling 2, 267–284 (1999)
59. Schwarz, U.M.: Scheduling related machines with failures (unpublished manuscript)
60. Schwarz, U.M.: Online scheduling on semi-related machines. Information Processing Letters 108(1), 38–40 (2008)
61. Schwarz, U.M., Diedrich, F.: Scheduling algorithms for random machine profiles (unpublished manuscript)
62. Schwarz, U.M., Diedrich, F.: A framework for scheduling with online availability. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 205–213. Springer, Heidelberg (2007)
63. Woltman, G., et al.: The great internet mersenne prime search, `http://www.mersenne.org/`

Iterative Compression for Exactly Solving NP-Hard Minimization Problems

Jiong Guo[*], Hannes Moser[**], and Rolf Niedermeier

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{guo,moser,niedermr}@minet.uni-jena.de

Abstract. We survey the conceptual framework and several applications of the iterative compression technique introduced in 2004 by Reed, Smith, and Vetta. This technique has proven very useful for achieving a number of recent breakthroughs in the development of fixed-parameter algorithms for NP-hard minimization problems. There is a clear potential for further applications as well as a further development of the technique itself. We describe several algorithmic results based on iterative compression and point out some challenges for future research.

1 Introduction

Until the year 2004, the parameterized complexity of several important NP-hard minimization problems was open. Then, Reed, Smith, and Vetta [43] introduced in a very short paper a new technique that is now called iterative compression. Meanwhile, based on this technique, a number of the mentioned open questions could be positively answered by giving corresponding fixed-parameter algorithms. To become more specific, let us consider the NP-complete Vertex Bipartization problem, where one is given an undirected graph $G = (V, E)$ and an integer $k \geq 0$, and the question is whether there is a set of at most k vertices such that their deletion leaves a bipartite graph. Due to iterative compression, now it is known that Vertex Bipartization can be solved in $O(3^k \cdot |V||E|)$ time [43,30]. In other words, Vertex Bipartization is fixed-parameter tractable with respect to the parameter k. Since then, similar breakthroughs have been achieved, for example, for the NP-complete problems Undirected Feedback Vertex Set [12,29,9], Directed Feedback Vertex Set [11], and Almost 2-Sat [42]. Here, we review the central ideas behind iterative compression and (some of) its applications. To this end, we choose the recently studied NP-complete Cluster Vertex Deletion problem [33] as our running example for exhibiting the "essentials" of iterative compression.

Notation. For a graph $G = (V, E)$ and a vertex set $S \subseteq V$, let $G[S]$ be the subgraph of G induced by S. A parameterized problem (I, k) is *fixed-parameter tractable* with respect to the parameter k if it can be solved in $f(k) \cdot \mathrm{poly}(|I|)$ time,

[*] Supported by the DFG, PALG, NI 369/8.

[**] Supported by the DFG, projects ITKO, NI 369/5 (part of the DFG-SPP 1126 "Algorithms on Large and Complex Networks") and AREG, NI 369/9.

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 65–80, 2009.

where I is the input instance (see [16,20,39]). Fixed-parameter tractability can be viewed as an alternative to polynomial-time approximability in dealing with NP-hard problems. An exact algorithm showing the fixed-parameter tractability of a parameterized problem is called *fixed-parameter algorithm.* In all graph problems that follow, n denotes the number of vertices and m denotes the number of edges.

2 Illustration of the Basic Technique

In the following, we exhibit the iterative compression technique by using the NP-complete Cluster Vertex Deletion (CVD) problem, arising in graph-based data clustering, as a running example.

> **Input:** An undirected graph $G = (V, E)$ and a nonnegative number k.
> **Question:** Is there a vertex subset $S \subseteq V$ with $|S| \leq k$ such that deleting all vertices in S from G results in a cluster graph, that is, a graph where every connected component forms a clique?

We present a simplified version of a more general algorithm [33] that solves the vertex-weighed variant of Cluster Vertex Deletion. Note that a graph is a cluster graph if and only if it contains no *induced* path of three vertices (referred to as P_3)[1].

The central idea of iterative compression is to employ a so-called *compression routine.* A *compression routine* is an algorithm that, given a problem instance and a corresponding solution, either calculates a smaller solution or proves that the given solution is of minimum size. Using a compression routine, one finds an optimal solution to a problem by inductively building up the problem structure and iteratively compressing intermediate solutions.

Example CVD. In the following, we call a solution for CVD a *cvd-set.* Here, the problem structure is built up vertex by vertex. We start with $V' = \emptyset$ and $S = \emptyset$; clearly, S is a cvd-set for $G[V']$. Iterating over all graph vertices, step by step we add one vertex $v \in V \setminus V'$ to both V' and S. Then S is still a cvd-set for $G[V']$. In each step, if $|S| > k$, then we try to find a smaller cvd-set for $G[V']$ by applying a compression routine. It takes the graph $G[V']$ and the cvd-set S for $G[V']$, and returns a smaller cvd-set for $G[V']$, or proves that S is optimal. If S is optimal, then we can conclude that G does not have a cvd-set of size at most k. Since eventually $V' = V$, we obtain a solution for G once the algorithm returns S. Note that almost all known applications of iterative compression on graph problems with vertex subsets as solutions essentially build up the graph in this way.[2]

The main point of the iterative compression technique is that if the compression routine is a fixed-parameter algorithm, then so is the whole algorithm. The main strength of iterative compression is that it allows to see the problem from a

[1] Three vertices $u, v, w \in V$ of a graph $G = (V, E)$ form an induced path if exactly two of the three edges $\{u, v\}$, $\{u, w\}$, $\{v, w\}$ are contained in E.

[2] An alternative example where the graph is built up edge by edge is given with the NP-complete Edge Bipartization problem [29].

different angle: The compression routine does not only have the problem instance as input, but also a solution, which carries valuable structural information on the input. Therefore, the design of a compression routine may be simpler than designing a fixed-parameter algorithm for the original problem.

While embedding the compression routine into the iteration framework is usually straightforward, finding the compression routine itself is not. It is not even clear that a compression routine with useful running time exists even when we already know a problem to be fixed-parameter tractable. Therefore, the art of iterative compression typically lies in the design of the compression routine. In many applications of iterative compression, the compression routine first exhaustively considers all possible intersection sets of the given solution and a potentially smaller solution; elements in the intersection can be "discarded" from the problem instance.[3] Then, the remaining task is to find a smaller *disjoint* solution.

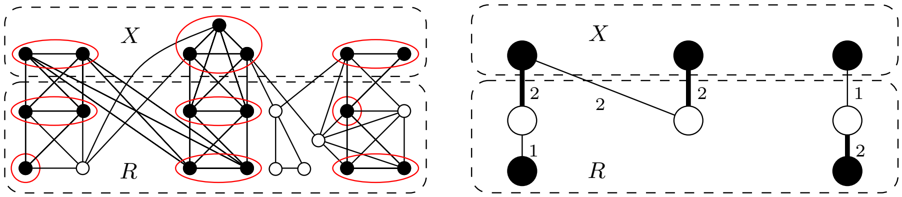
Example CVD continued. For Cluster Vertex Deletion the compression routine works as follows. Consider a smaller cvd-set S' as a modification of the larger cvd-set S for the graph $G = (V, E)$. This modification retains some vertices $Y \subsetneq S$ as part of the solution set (that is, the vertices to be deleted), while the other vertices $X := S \setminus Y$ are replaced by new vertices from $V \setminus S$. The idea is to try by brute force all $2^{|S|} - 1$ nontrivial partitions of S into these two sets Y and X. For each such partition, the vertices from Y are immediately deleted, since we already decided to take them into the cvd-set. In the resulting instance $G' = (V', E') := G[V \setminus Y]$, it remains to find a smaller cvd-set that is disjoint from X. This turns out to be a much easier task than finding a cvd-set in general; in fact, it can be done in polynomial time using data reduction and maximum matching.

The idea to try by brute force all nontrivial partitions of a given solution S into two sets Y and X is applied for all the problems in this survey; thus, we always describe the corresponding compression routines by showing how to compute a smaller *disjoint* solution.

Compression for CVD. Recall that X is a cvd-set for G', and the task is to find a smaller cvd-set for G' disjoint from X. First, we discard partitions where X does not induce a cluster graph; these partitions cannot lead to a solution since we fixed that none of the vertices in X would be deleted. Further, the set $R := V' \setminus X$ also induces a cluster graph since $R = V \setminus S$ and S is a cvd-set. Therefore, the following computational problem remains:

> CVD Compression
> **Instance:** An undirected graph $G = (V, E)$ and a vertex set $X \subseteq V$ such that $G[X]$ and $G[V \setminus X]$ are cluster graphs.
> **Task:** Find a vertex set $X' \subseteq V \setminus X$ such that $G[V \setminus X']$ is a cluster graph and $|X'| < |X|$.

[3] It depends on the concrete problem of how we "discard" these elements. For instance, for a vertex deletion problem we may just remove the corresponding vertices.



(a) CVD Compression instance (b) The assignment problem

Fig. 1. (a) Data reduction in the compression routine. The white vertices in the input instance are removed by the three data reduction rules. Each group of encircled vertices corresponds to a vertex in the assignment problem (b). If a group of encircled vertices is in X or if no vertex in this group has a neighbor in X, then the corresponding vertex is black, otherwise, it is white.

An example for a CVD Compression instance is shown in Figure 1a. In a first step, this instance can be simplified by a series of simple data reduction rules; their correctness is easy to see [33]:

1. Delete all vertices in $R := V \setminus X$ that are adjacent to more than one clique in $G[X]$.
2. Delete all vertices in R that are adjacent to some, but not all vertices of a clique in $G[X]$.
3. Remove connected components that are cliques.

After these data reduction rules have been exhaustively applied, the instance is much simplified: In each clique of $G[R]$, we can divide the vertices into equivalence classes according to their neighborhood in X; each class then contains either vertices adjacent to all vertices of a particular clique in $G[X]$, or the vertices adjacent to no vertex in X (see Figure 1a). This classification is useful because of the following.

Lemma 1. *If there exists a solution for* CVD Compression*, then in the cluster graph resulting by this solution, for each clique in* $G[R]$ *the vertices of at most one equivalence class are present.*

Proof. Clearly, inside a clique, it is never useful to delete only some, but not all vertices of an equivalence class, since if that led to a solution, we could always re-add the deleted vertices without introducing new induced P_3's (which are the forbidden substructure characterizing cluster graphs). Further, assume that for a clique C in $G[R]$ the vertices of two equivalence classes are present. Let $u \in C$ and $v \in C$ be a vertex from each equivalence class, respectively. Since u and v are in different equivalence classes, they must have a different neighborhood with respect to the cliques in $G[X]$. Assume without loss of generality that v is adjacent to all vertices of a clique C' in $G[X]$. Since u is in an other equivalence class than v, u is not adjacent to any vertex of C'. Let $w \in C'$. The path uvw forms an induced P_3, contradicting our assumption. □

Due to Lemma 1, the remaining task for solving CVD Compression is to assign each clique in $G[R]$ to one of its equivalence classes (corresponding to the preservation of this class, and the deletion of all vertices from the other classes within the clique) or to nothing (corresponding to the complete deletion of the clique). However, we cannot do this independently for each clique; we must not choose two classes from different cliques in $G[R]$ such that these two classes are adjacent to the same clique in $G[X]$ since that would create an induced P_3. This assignment problem can be modelled as a weighted bipartite matching problem in an auxiliary graph H, where each edge corresponds to a possible choice. The graph H is constructed as follows (see Figure 1b):

1. Add a vertex for every clique in $G[R]$ (white vertices).
2. Add a vertex for every clique in $G[X]$ (black vertices in X).
3. For a clique C_X in $G[X]$ and a clique C_R in $G[R]$, add an edge between the vertex for C_X and the vertex for C_R if there is an equivalence class in C_R containing a vertex adjacent to a vertex in C_X. This edge corresponds to choosing this class for C_R and one assigns the number of vertices in this class as its weight.
4. Add a vertex for each class in a clique C_R that is not adjacent to a clique in $G[X]$ (black vertices outside X), and connect it to the vertex representing C_R. Again, this edge corresponds to choosing this class for C_R and is weighted with the number of vertices in this class.

Since we only added edges between black and white vertices, H is bipartite. The task is now to find a *maximum-weight bipartite matching*, that is, a set of edges of maximum weight where no two edges have an endpoint in common. To solve this matching instance, we can use an algorithm for integer weighted matching with a maximum weight of n [24], yielding a running time of $O(m\sqrt{n}\log n)$. If we apply the data reduction rules in their given order, we can execute them in $O(m)$ time. Thus, we can solve CVD Compression in $O(m\sqrt{n}\log n)$ time. The number of vertices in an intermediary solution S to be compressed is bounded from above by $k+1$, because any such S consists of a size-at-most-k solution for a subgraph of G plus a single vertex. In one iteration step, CVD Compression is thus solved $O(2^k)$ times, and there are n iteration steps, yielding a total running time of $O(2^k \cdot mn^{3/2}\log n)$. With some tricks based on problem kernelization, which will not be explained here, this can be further improved to $O(2^k k^6 \log k + nm)$.

Theorem 1 ([33]). Cluster Vertex Deletion *can be solved within a running time of* $O(2^k k^6 \log k + nm)$.

This iterative compression approach combined with matching techniques also works for the vertex-weighted version of Cluster Vertex Deletion, yielding an algorithm with a running time of $O(2^k k^9 + nm)$ [33].

As we have seen for Cluster Vertex Deletion, the iterative part of building up the graph vertex by vertex is relatively simple, but the problem-specific compression step of applying data reduction and matching techniques is more involved. In many applications of iterative compression, the iterative part is carried out in the same manner, and data reduction techniques often play an important role for the compression step.

3 Some Breakthroughs Due to Iterative Compression

In this section, we survey recent breakthroughs in parameterized algorithmics that all are based on a sophisticated use of iterative compression. We mainly focus on describing the central ideas behind the respective compression routines.

3.1 Graph Bipartization

Definition and History. The NP-complete Graph Bipartization problem, also known as Vertex Bipartization, Maximum Bipartite Induced Subgraph, or Odd Cycle Transversal, is defined as follows.

> **Input:** An undirected graph $G = (V, E)$ and a nonnegative number k.
> **Question:** Is there a vertex subset $S \subseteq V$ with $|S| \leq k$ such that $G[V \backslash S]$ is bipartite?

Graph Bipartization has applications ranging from VLSI design to computational biology (see, e.g., [35,45]). There is a polynomial-time approximation with a factor of $O(\log n)$ [25], and there is an exact algorithm running in $O(1.62^n)$ time [41]. Mahajan and Raman [36] explicitly mentioned the fixed-parameter tractability of Graph Bipartization with respect to the parameter k as an open problem, which was settled by Reed et al. [43], thereby introducing the iterative compression technique. The running time of their approach is stated as $O(4^k \cdot knm)$, but with a better analysis and an algorithmic trick, it can been improved to $O(3^k \cdot nm)$ [30,31].

Compression Routine. In the following, we outline the basic idea of the compression routine due to Reed et al. [43]. We have as input an undirected graph $G = (V, E)$ and a vertex set $X \subseteq V$ such that $G[V \setminus X]$ is bipartite. The question is whether there exists a set X' with $X' \subseteq V \backslash X$ and $|X'| < |X|$ such that $G[V \backslash X']$ is bipartite. Let I_1 and I_2 be the two partite sets of $G[V \setminus X]$. The idea is to construct an auxiliary graph corresponding to G: Replace every vertex $v \in X$ by two vertices v_1, v_2, deleting all edges incident to v. For every edge vw in G with $w \in V \setminus X$, we add the edge v_1w if $w \in I_2$ and v_2w if $w \in I_1$. For every edge vw with $w \in X$, we arbitrarily add the edge v_1w_2 or the edge v_2w_1. See Figure 2 for an example. The idea behind that construction is to enforce that if there exists an odd cycle (a cycle with an odd number of edges) in G going through a vertex a (e.g., the cycle (a, b, e) in Figure 2), then there exists a path from a_1 to a_2 in the auxiliary graph. Note that a path from a_1 to a_2 in the auxiliary graph only implies some odd cycle in G (but not necessarily containing a). Let $N_X := \{v_1, v_2 \mid v \in X\}$ be the set of the newly introduced vertices. A partition (A, B) of N_X is *valid* if for each vertex pair v_1, v_2 either $v_1 \in A$ and $v_2 \in B$ or $v_1 \in B$ and $v_2 \in A$. The intuitive idea of the compression routine is to try to find a valid partition (A, B) such that all paths between A and B can be obstructed by less than $|X|$ vertices. These vertices also obstruct every odd cycle in G (by the construction of the auxiliary graph). The following lemma, which is a variation of Lemma 1 in [43] and will not be proven here, shows that this approach is correct.

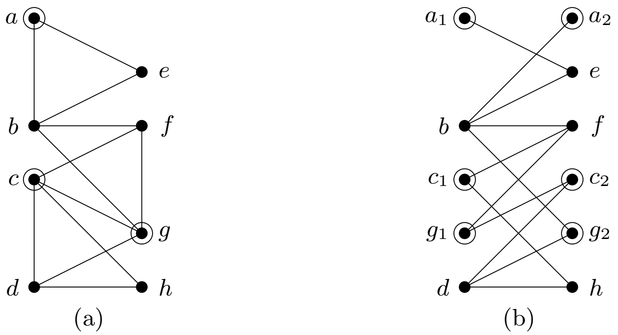


Fig. 2. (a) G with solution (encircled vertices); (b) Corresponding auxiliary graph

Lemma 2. *If for any valid partition (A, B) of N_X there are $|X|$ vertex-disjoint paths from A to B in the corresponding auxiliary graph, then there is no solution X' with $X' \subseteq V \setminus X$ and $|X'| < |X|$.*

Using Lemma 2, we can enumerate in $2^{|X|}$ steps all possible valid partitions (A, B) of N_X, and compute a vertex cut between (A, B) using maximum flow techniques. If the vertex cut contains less than $|X|$ vertices, then we return it as the new solution X'. The maximum flow can be computed in $O(km)$ time [23], and since $|X| \leq k+1$ the overall running time of the compression routine is $O(2^k \cdot km)$. With an algorithmic trick, which "recycles" the flow networks for each maximum flow problem, one can get rid of the factor k in the running time [30]. Together with the iteration and the partitioning needed for the compression routine, an improved analysis, not described here, yields the following.

Theorem 2 ([43,30]). Graph Bipartization *can be solved within a running time of $O(3^k \cdot nm)$.*

Further Remarks. There exists another approach using vertex colorings to describe the Graph Bipartization algorithm [31]. The Graph Bipartization algorithm has also been heuristically improved and implemented [30,31].[4] The experiments on data from computational biology show that iterative compression can outperform other methods by orders of magnitude. For example, an instance originating from computational biology with 102 vertices and 307 edges can be solved in 6248 seconds with an ILP approach, whereas an iterative compression approach runs in 0.79 seconds, which can be further improved by algorithmic tricks [30]. The iterative compression approach also works for the variant of making a given graph bipartite by at most k edge deletions, and yields an algorithm with a running time of $O(2^k \cdot m^2)$ [29]. This algorithm can be used for Signed Graph Balancing, where experiments show that this approach

[4] The source code of the Graph Bipartization solver and corresponding test data are available from `http://theinf1.informatik.uni-jena.de/occ`

has about the same running time as approximation algorithms, while producing exact solutions [32][5].

The Almost 2-SAT problem is a generalization of Graph Bipartization. It asks whether it is possible to delete at most k clauses from a Boolean formula in conjunctive normal form with at most two literals per clause such that the remaining formula becomes satisfiable. Very recently, Razgon and O'Sullivan [42] showed that Almost 2-SAT can be solved in $O(15^k k \cdot m^3)$ time, thus proving it to be fixed-parameter tractable with respect to parameter k, which was an open question stated by Mahajan and Raman [36]. This result also implies that Vertex Cover parameterized above the size of a maximum matching M, that is, the task to find a vertex cover of size at most $|M| + k$ ("above guarantee parameterization") [38], is fixed-parameter tractable with respect to the parameter k, because it can be transformed to Almost 2-SAT [42] in $f(k) \cdot \text{poly}(n)$ time.

3.2 Undirected Feedback Vertex Set

Definition and History. The NP-complete Undirected Feedback Vertex Set (UFVS) problem is defined as follows.

> **Input:** An undirected graph $G = (V, E)$ and a nonnegative number k.
> **Question:** Is there a *feedback vertex set* (fvs) $S \subseteq V$ with $|S| \leq k$, that is, a set S whose deletion from G results in a forest?

UFVS has found applications in many fields, including deadlock prevention, program verification, and Bayesian inference [19]. UFVS can be approximated to a factor of 2 in polynomial time [1,4]. There is a simple and elegant randomized algorithm [3] that solves UFVS in $O(c4^k k \cdot n)$ time by finding a feedback vertex set of size k with probability at least $1 - (1 - 4^{-k})^{c4^k}$ for an arbitrary constant c. A recent exact algorithm for UFVS has running time of $O(1.7548^n)$ [22]. As to deterministic fixed-parameter algorithms, Bodlaender [7] and Downey and Fellows [15] were the first to show that the problem is fixed-parameter tractable. In 2005, Dehne et al. [12] and Guo et al. [29] independently provided the first algorithms, based on iterative compression, with running times of $O(c^k \cdot nm)$ with $c \approx 11$. Finally, Chen et al. [9], also employing iterative compression, improved the constant c to 5.

Compression Routine. In the following, we briefly describe the basic idea behind the compression routine due to Chen et al. [9]. Herein, we have as input an undirected graph $G = (V, E)$ and an fvs X. The question is whether there exists an fvs X' with $|X'| < |X|$ and $X' \subseteq V \setminus X$. Observe that G can be divided into two forests: The induced subgraph $G[V \setminus X]$ is clearly acyclic. Moreover, $G[X]$ is also a forest consisting of at most $|X|$ trees; otherwise, there would not exist an fvs X' with $X \cap X' = \emptyset$. Based on this observation, the key idea of the

[5] The source code of the Signed Graph Balancing solver is available from `http://theinf1.informatik.uni-jena.de/bsg`

compression routine is to merge these two forests into one by deleting as few as possible vertices from $V \setminus X$; that is, it considers every vertex $v \in V \setminus X$ and tries to move v from $V \setminus X$ to X without introducing a cycle in $G[X]$. Since a solution X' must be disjoint from X, we have to include v into X' if its addition to X creates a cycle. If there is no cycle created by adding v to X, then make a trivial branch into two subcases, namely, (1) keeping X unchanged, adding v to X', and deleting v from G or (2) extending X by v. In the latter case, we assume that v is not part of a solution. More specifically, the compression routine follows a search tree strategy which distinguishes the following cases: First, if there is a vertex $v \in V \setminus X$ that has two neighbors from the same tree in $G[X]$, then add v to X' and delete v from G. This is clearly correct, since moving v to X introduces cycles. Second, if the first case does not apply and there is a vertex $v \in V \setminus X$ that has at least two neighbors in X, then we know that these neighbors are from different trees in $G[X]$ and, thus, $G[X \cup \{v\}]$ is acyclic; we branch the search into two subcases as described above. Finally, if the first two cases do not apply, then all vertices in $V \setminus X$ have at most one neighbor in X. We apply the following bypassing reduction rule to the leaves of the forest $G[V \setminus X]$, until one the first two cases applies or $V \setminus X = \emptyset$. We say that a graph G' is obtained from G by *bypassing* a degree-2 vertex v in G if G' is obtained by first removing v and then adding a new edge between its two neighbors. The following data reduction rule is trivially correct, because deleting degree-2 vertices to destroy cycles is never the only best solution.

Bypassing reduction rule: Bypass all degree-2 vertices in G.

To analyze the running time of the compression routine, we have to bound the search tree size. Since only the second case causes a branching into two subcases, it remains to upper-bound the number of the applications of the second case. We claim that this number is bounded by $j + l$, where j denotes the size of the input set X and l denotes the number of trees in $G[X]$. In the first subcase of the branching caused by the second case, we add a vertex to X', which can only happen at most $j - 1$ times, because we search an X' with $|X'| < |X|$. The second subcase adds vertex v to X. Since v has at least two neighbors in X which are from different trees in $G[X]$ (the precondition of the second case), the addition of v to X causes a merge of at least two trees and, thus, decreases the number of trees in $G[X]$. Clearly, this can be done at most l times. Altogether, in the branchings caused by the second case, we either decrease the number of trees in $G[X]$ or add a vertex to X'. The second case can apply less than $j + l$ times. Since $j \leq k + 1$ and, thus, $G[X]$ has at most $k + 1$ trees, the search tree has size $O(2^{2k})$, giving an overall running time of the compression routine of $O(4^k \cdot n^2)$, where we need $O(n^2)$ time to apply the bypassing rule and to check the applicability of the three cases. Together with the iteration and the partitioning needed for the compression routine, one arrives at the following theorem.

Theorem 3 ([9]). Undirected Feedback Vertex Set *can be solved within a running time of* $O(5^k k \cdot n^2)$.

Further Remarks. The randomized algorithm by Becker et al. [3] is so elegant and simple such that it seems to be the current method of choice for practically computing an optimal undirected feedback vertex set; the current best iterative compression approach is still slower and more complicated to implement. The currently best problem kernel for UFVS has $O(k^3)$ vertices and edges [8]. Accepting a worse exponential base c, there is also a deterministic “linear-time FPT” algorithm for UFVS running in $O(c^k \cdot (m+n))$ time [29].

3.3 Directed Feedback Vertex Set

Definition and History. The NP-complete Directed Feedback Vertex Set (DFVS) problem is defined as follows.

> **Input:** A directed graph $G = (V, E)$ and a nonnegative number k.
> **Question:** Is there a *feedback vertex set* (fvs) $S \subseteq V$ with $|S| \leq k$, that is, a vertex set S whose deletion from G results in a graph with no directed cycles?

DFVS has various applications such as deadlock prevention in operating systems and database systems, circuit testing, voting systems, and computational biology [19]. DFVS can be approximated to a factor of $O(\log n \log \log n)$ in polynomial time [17]. For about 15 years, it has been repeatedly stated as an open question whether or not DFVS is fixed-parameter tractable with respect to the parameter k. Very recently, Chen et al. [11] gave a fixed-parameter algorithm with a running time of $O(k!4^k k^3 \cdot n^4)$, thus answering this open question. Comparing with the result for UFVS, the particular difficulty in case of DFVS is given by the fact that destroying all directed cycles not only leaves trees as in the undirected case but leaves directed acyclic graphs, much more complicated structures than trees. Thus, the result for UFVS might be interpreted as a first “necessary” intellectual step before the result for DFVS was within reach.

Compression Routine. In the following, we briefly describe the idea behind the compression routine due to Chen et al. [11]. Herein, we have as input a directed graph $G = (V, E)$ and an fvs X. The question is whether there exists an fvs X' with $|X'| < |X|$ and $X' \subseteq V \setminus X$. Consider an fvs X' disjoint from X in G. Since there is no directed cycle in $G[V \setminus X']$, there must be a vertex $u \in X$ such that for each vertex $v \in X$ there is no directed path from u to v in $G[V \setminus X']$ (this can be easily seen: if there is a path from every vertex in X to at least one other vertex in X, following such paths from vertex to vertex will end up visiting an already visited vertex, yielding a directed cycle in $G[V \setminus X']$). A repeated application of this argument shows that there is an ordering $u_1, \ldots, u_{|X|}$ of the vertices in X such that there is no directed path in $G[V \setminus X']$ from u_i to u_j if $i \geq j$. To find the set X', the idea is now to try all $|X|!$ orderings of the vertices in X, and, for each ordering $u_1, \ldots, u_{|X|}$, we compute X' such that there is no directed path from u_i to u_j if $i \geq j$ in $G[V \setminus X']$. To this end, we split each vertex $u_i \in X$ into two vertices s_i and t_i such that all outgoing edges of u_i are incident to s_i and all incoming edges of u_i are incident to t_i. Then, the task is to find a vertex separator X' between the vertices $s_1, \ldots, s_{|X|}$ and $t_1, \ldots, t_{|X|}$ such that there is no path from s_i to t_j if $i \geq j$. This task can be solved by an algorithm for the Skew Separator problem, which is defined as follows.

> **Instance:** A directed graph $G = (V, E)$, a parameter k, and pairwise disjoint vertex subsets $S_1, \ldots, S_l$, $T_1, \ldots, T_l$ such that there is no edge going into a set S_i for $1 \leq i \leq l$ and such that there is no edge going out from a set T_i for $1 \leq i \leq l$.
> **Task:** Find a *skew separator* $X' \subseteq V$ for G with $|X'| \leq k$, that is, a vertex set X' such that there is no directed path from any vertex in S_i to any vertex in T_j in $G[V \setminus X']$ if $i \geq j$.

Skew Separator can be solved by a clever branching strategy in a running time of $O(4^k k \cdot n^3)$ [11]. The corresponding algorithm is a directed variant of an algorithm for the Minimum Node Multiway Cut problem on undirected graphs [10], where the task is, given a graph G, a parameter k, and vertex-disjoint terminal vertex sets, to delete at most k vertices such that there is no path between any two vertices of two different terminal sets. Using the algorithm for Skew Separator, one solves the task to find a vertex separator X' by setting $S_i := \{s_i\}$ and $T_i := \{t_i\}$ for all $1 \leq i \leq |X|$ and solving Skew Separator in $O(4^k k \cdot n^3)$ time.

Next, we analyze the running time of the whole iterative compression approach. There are $|X|! \leq (k+1)!$ orderings of the vertices in X, thus the algorithm for Skew Separator is called at most $(k+1)!$ times. Altogether, the compression routine runs in $O(k! 4^k k^2 \cdot n^3)$ time. With some relatively simple analysis, it is possible to show that together with the iteration and the partitioning needed for the compression routine we arrive at the following theorem.

Theorem 4 ([11]). Directed Feedback Vertex Set *can be solved within a running time of* $O(k! 4^k k^3 \cdot n^4)$.

Further Remarks. The above algorithm for DFVS finds a direct application in Kemeny voting systems for the problem of computing the Kemeny score in case of incomplete votes [5], and for the dual of the Longest Compatible Sequence problem [28]. Moreover, in the special case of DFVS where the input graph is restricted to be a tournament (that is, a directed graph whose “underlying” undirected graph is complete), an iterative compression approach yields an algorithm with a running time of $O(2^k \cdot n^2(\log\log n + k))$ [14].

4 Discussion and Future Challenges

Discussion. Iterative compression is an algorithm design principle based on induction. This elegant and simple approach is appealing and invites for further applications and extensions. On the one hand, iterative compression may strongly benefit from a combination with kernelization and corresponding data reduction rules (see, for instance, the case of Undirected Feedback Vertex Set in Subsection 3.2), and, on the other hand, has also been employed for achieving

(Turing) kernelization results [13]. Iterative compression may also be used for enumerating all minimal solutions of size at most k in FPT time, as the example Undirected Feedback Vertex Set shows [29]. In many applications of iterative compression, one often occurring running time bottleneck is that all partitions of a size-k solution have to be considered in order to find a smaller-size solution (see Section 2). This typically incurs an additional running time factor of 2^k. However, in some cases this factor can be avoided by proving a guarantee that the improved solution can without loss of generality be assumed to be disjoint from the old solution. An example for this is given by the Edge Bipartization problem [29]. It appears to be fruitful to combine iterative compression with efficient polynomial-time approximation algorithms. The idea is that the compression routine may start right away with the solution provided by a constant-factor approximation, saving time otherwise needed for the iteration procedure [29]. Iterative compression is not only a tool for purely theoretical algorithm design but has already proven practical usefulness in few experimental studies [30,31,32]. One reason for this practical success also lies in the flexibility of the use of the iterative compression routine [34]. It can be started on any suboptimal initial solution to improve this solution. Moreover, it can be stopped whenever the found solution is good enough. Finally, it should be mentioned that the known applications of iterative compression have a strong focus on graph modification problems with the goal to generate graphs with hereditary properties. It would be interesting to see more applications of iterative compression outside this scenario, the case of Almost 2-Sat [42] meaning a first step.

First experimental results for iterative compression-based algorithms appear quite encouraging. An implementation of the Graph Bipartization algorithm, improved by heuristics, can solve all instances from a testbed from computational biology within minutes, whereas established methods are only able to solve about half of the instances within reasonable time. For instance, in case of Graph Bipartization instances of sizes up to 296 vertices and 1620 edges could be solved within less than 4 minutes [30]. Further, an iterative compression based approach for the Balanced Subgraph problem, which generalizes Edge Bipartization, is able to find optimal solutions to instances for which previously only approximate solutions could be given [32].

So far, the list of successful applications of iterative compression is impressive but still clear in its range. Future research has to determine how far-ranging applications of iterative compression can be found. Clearly, this survey only sketched a few results achieved by iterative compression. One further sophisticated and technically fairly demanding application of iterative compression has been developed by Marx [37] to show that the problem to delete a minimum number of vertices in order to make a graph chordal is fixed-parameter tractable.

Future Challenges. As already mentioned, there are close connections between iterative compression and approximation algorithms as well as kernelization. In both cases, there seems to be room for further fruitful explorations. A recent paper makes a first attempt in linking iterative compression with exact algorithms [21]. Another conceptually related field of research is that of reoptimization [6]. There, the scenario is that one is given an instance of an optimization problem together with an optimal solution, and one wants to find a "high-quality" solution for a locally modified instance. Finding useful connections between this scenario and the compression scenario would be highly interesting. Similarly, exploring connections to the paradigm of local search might be promising. In addition, it would be interesting to investigate the connections between augmentation problems in the context of 0/1-integer linear programming and iterative compression. In a nutshell, in these augmentation problem one is given a vector $c \in \mathbb{Z}^n$, a feasible solution x over $\{0, 1\}^n$, and the task is to find out whether there is another feasible solution y such that $c \cdot y > c \cdot x$, or to assert that no such y exists [44].

Another general issue of future research may address the inductive structure of iterative compression. So far, all applications of iterative compression have a pretty simple inductive structure, and there is no reason to believe that more sophisticated structures might not be helpful. Finally, the experiences concerning algorithm engineering results for compression-based methods are still very limited and need to be extended.

We end this survey with a list of few open questions relating to concrete computational problems. The solution of each of these would mean progress for iterative compression and parameterized algorithmics at large.

- Directed Feedback Vertex Set now having been classified in terms of parameterized complexity analysis using iterative compression, a natural next step seems to be to attack the "more general" problems of Shortest Common Supersequence and Smallest Common Supertree (see [18] for definitions) in an analogous way. The parameterized complexity of these problems is unsettled.
- Directed Feedback Vertex Set restricted to tournament graphs is solvable in $2^k \cdot n^{O(1)}$ time using iterative compression [14]. An analogous result for Directed Feedback Edge Set restricted to tournaments is missing.
- Edge Clique Cover is fixed-parameter tractable but the only known way to achieve this is based on a simple, exponential-size kernelization [27]. Can iterative compression lead to a more efficient fixed-parameter algorithm for Edge Clique Cover?
- Can iterative compression be used to show the fixed-parameter tractability of the Correlation Clustering problem [2], a generalization of the fixed-parameter tractable Cluster Editing problem [26]?
- Contig Scaffolding problems appear in the analysis of genomic data [40]. Again, iterative compression here might become a door opener for efficient fixed-parameter algorithms, which so far are not known for these problems.

Acknowledgement. We are grateful to Christian Komusiewicz, Matthias Müller-Hannemann, and Siavash Vahdati Daneshmand for constructive comments improving the presentation of this paper.

References

1. Bafna, V., Berman, P., Fujito, T.: A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM J. Discrete Math. 3(2), 289–297 (1999)
2. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Mach. Learn. 56(1-3), 89–113 (2004)
3. Becker, A., Bar-Yehuda, R., Geiger, D.: Randomized algorithms for the Loop Cutset problem. J. Artificial Intelligence Res. 12, 219–234 (2000)
4. Becker, A., Geiger, D.: Approximation algorithms for the Loop Cutset problem. In: Proc. 10th UAI, pp. 60–68. Morgan Kaufmann, San Francisco (1994)
5. Betzler, N., Fellows, M.R., Guo, J., Niedermeier, R., Rosamond, F.A.: Fixed-parameter algorithms for Kemeny scores. In: Fleischer, R., Xu, J. (eds.) AAIM 2008. LNCS, vol. 5034, pp. 60–71. Springer, Heidelberg (2008)
6. Böckenhauer, H.-J., Hromkovic, J., Mömke, T., Widmayer, P.: On the hardness of reoptimization. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 50–65. Springer, Heidelberg (2008)
7. Bodlaender, H.L.: On disjoint cycles. Int. J. Found. Computer Science 5, 59–68 (1994)
8. Bodlaender, H.L.: A cubic kernel for feedback vertex set. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 320–331. Springer, Heidelberg (2007)
9. Chen, J., Fomin, F.V., Liu, Y., Lu, S., Villanger, Y.: Improved algorithms for the feedback vertex set problems. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 422–433. Springer, Heidelberg (2007)
10. Chen, J., Liu, Y., Lu, S.: An improved parameterized algorithm for the minimum node multiway cut problem. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 495–506. Springer, Heidelberg (2007)
11. Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I.: A fixed-parameter algorithm for the directed feedback vertex set problem. In: Proc. 40th STOC, pp. 177–186. ACM Press, New York (2008)
12. Dehne, F.K.H.A., Fellows, M.R., Langston, M.A., Rosamond, F.A., Stevens, K.: An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. Theory Comput. Syst. 41(3), 479–492 (2007)
13. Dehne, F.K.H.A., Fellows, M.R., Rosamond, F.A., Shaw, P.: Greedy localization, iterative compression, and modeled crown reductions: New FPT techniques, an improved algorithm for set splitting, and a novel $2k$ kernelization for Vertex Cover. In: Downey, R.G., Fellows, M.R., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 271–280. Springer, Heidelberg (2004)
14. Dom, M., Guo, J., Hüffner, F., Niedermeier, R., Truß, A.: Fixed-parameter tractability results for feedback set problems in tournaments. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) CIAC 2006. LNCS, vol. 3998, pp. 320–331. Springer, Heidelberg (2006)
15. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness. Congr. Numer. 87, 161–187 (1992)
16. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
17. Even, G., Naor, J., Schieber, B., Sudan, M.: Approximating minimum feedback sets and multicuts in directed graphs. Algorithmica 20(2), 151–174 (1998)
18. Fellows, M.R., Hallett, M.T., Stege, U.: Analogs & duals of the MAST problem for sequences & trees. J. Algorithms 49(1), 192–216 (2003)

19. Festa, P., Pardalos, P.M., Resende, M.G.C.: Feedback set problems. In: Du, D.-Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization, supplement vol. A, pp. 209–259. Kluwer Academic Publishers, Dordrecht (1999)
20. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
21. Fomin, F., Gaspers, S., Kratsch, D., Liedloff, M., Saurabh, S.: Iterative compression and exact algorithms. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 335–346. Springer, Heidelberg (2008)
22. Fomin, F.V., Gaspers, S., Pyatkin, A.V.: Finding a minimum feedback vertex set in time $O(1.7548^n)$. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 184–191. Springer, Heidelberg (2006)
23. Fulkerson, D.R., Ford Jr., L.R.: Maximal flow through a network. Canad. J. Math. 8, 399–404 (1956)
24. Gabow, H.N., Tarjan, R.E.: Faster scaling algorithms for network problems. SIAM J. Comput. 18(5), 1013–1036 (1989)
25. Garg, N., Vazirani, V.V., Yannakakis, M.: Multiway cuts in directed and node weighted graphs. In: Shamir, E., Abiteboul, S. (eds.) ICALP 1994. LNCS, vol. 820, pp. 487–498. Springer, Heidelberg (1994)
26. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: Exact algorithms for clique generation. Theory Comput. Syst. 38(4), 373–392 (2005)
27. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Data reduction, exact, and heuristic algorithms for clique cover. In: Proc. 8th ALENEX, pp. 86–94. SIAM, Philadelphia (2006); Journal version to appear under the title Data reduction and exact algorithms for clique cover. ACM J. Exp. Algorithmics
28. Guillemot, S.: Parameterized complexity and approximability of the SLCS problem. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 115–128. Springer, Heidelberg (2008)
29. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. J. Comput. System Sci. 72(8), 1386–1396 (2006)
30. Hüffner, F.: Algorithm engineering for optimal graph bipartization. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 240–252. Springer, Heidelberg (2005)
31. Hüffner, F.: Algorithms and Experiments for Parameterized Approaches to Hard Graph Problems. Ph.D thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena (2007)
32. Hüffner, F., Betzler, N., Niedermeier, R.: Optimal edge deletions for signed graph balancing. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 297–310. Springer, Heidelberg (2007)
33. Hüffner, F., Komusiewicz, C., Moser, H., Niedermeier, R.: Fixed-parameter algorithms for cluster vertex deletion. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 711–722. Springer, Heidelberg (2008)
34. Hüffner, F., Niedermeier, R., Wernicke, S.: Techniques for practical fixed-parameter algorithms. The Computer Journal 51(1), 7–25 (2008)
35. Kahng, A.B., Vaya, S., Zelikovsky, A.: New graph bipartizations for double-exposure, bright field alternating phase-shift mask layout. In: Proc. Asia and South Pacific Design Automation Conference, pp. 133–138. ACM Press, New York (2001)
36. Mahajan, M., Raman, V.: Parameterizing above guaranteed values: MaxSat and MaxCut. J. Algorithms 31(2), 335–354 (1999)

37. Marx, D.: Chordal deletion is fixed-parameter tractable. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 37–48. Springer, Heidelberg (2006)
38. Mishra, S., Raman, V., Saurabh, S., Sikdar, S., Subramanian, C.R.: The complexity of finding subgraphs whose matching number equals the vertex cover number. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 268–279. Springer, Heidelberg (2007)
39. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
40. Pop, M., Kosack, D.S., Salzberg, S.L.: Hierarchical scaffolding with Bambus. Genome Research 14(1), 149–159 (2004)
41. Raman, V., Saurabh, S., Sikdar, S.: Efficient exact algorithms through enumerating maximal independent sets and other techniques. Theory Comput. Syst. 41(3), 563–587 (2007)
42. Razgon, I., O'Sullivan, B.: Almost 2-SAT is fixed-parameter tractable. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 551–562. Springer, Heidelberg (2008)
43. Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. Oper. Res. Lett. 32(4), 299–301 (2004)
44. Schulz, A.S., Weismantel, R., Ziegler, G.M.: 0/1-integer programming: Optimization and augmentation are equivalent. In: Spirakis, P.G. (ed.) ESA 1995. LNCS, vol. 473–483, pp. 473–483. Springer, Heidelberg (1995)
45. Zhang, X.-S., Wang, R.-S., Wu, L.-Y., Chen, L.: Models and algorithms for haplotyping problem. Current Bioinformatics 1(1), 104–114 (2006)

Approaches to the Steiner Problem in Networks*

Tobias Polzin[1] and Siavash Vahdati-Daneshmand[2]

[1] HaCon Ingenieurgesellschaft mbH, Hannover, Germany
[2] Theoretische Informatik, Universität Mannheim, Germany
vahdati@informatik.uni-mannheim.de

Abstract. The Steiner problem in networks is the problem of connecting a set of required vertices in a weighted graph at minimum cost. This is a classical $\mathcal{NP}$-hard problem and a fundamental problem in network design with many practical applications.

We approach this problem by various means: Relaxations, which relax the feasibility constraints, to get close to an optimal solution; heuristics to find good, but not necessarily optimal solutions; and reductions to simplify problem instances without abandoning the optimal solution. We have integrated these components into an exact algorithm that has achieved outstanding results in practice.

In this article, we first provide a brief overview on the main algorithmic developments related to our work on this problem, citing our and others' (already published) works. Then we focus on some central concepts, presenting detailed results on selected topics that offer special insight and potential for further improvement.

1 Introduction

The Steiner problem in networks is the problem of connecting a set of required vertices in a weighted graph at minimum cost. This is a classical $\mathcal{NP}$-hard problem and a fundamental problem in network design with many applications.

The Steiner problem is one of the best-known and most-studied problems in (combinatorial) optimization. For the Steiner problem in networks, as we study it here, there are more than a hundred publications; this number grows to several hundreds if one also considers variants of this problem. Many books have been written on this problem, for example [6,9,10,16,30,36]. The study of the Steiner problem is motivated by its central role in network design and by its numerous practical applications [6,16,21]. New theoretical results on this problem lead directly to corresponding results for many other problems and the analysis methods can be used for related problems [17,20]. From an algorithmic point of view, classical $\mathcal{NP}$-hard combinatorial optimization problems like the traveling salesman problem or the Steiner problem have served as "engines of discovery" for new methods that can also be applied to other problems.

Why do we try to solve such problems exactly? Of course, we cannot deny that the challenge is tempting: These prominent $\mathcal{NP}$-hard problems have attracted

* Supported by the DFG, project NetDesign, KR 1521/6 (part of the DFG-SPP 1126).

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 81–103, 2009.

researchers as test environments for their methods for decades. As a consequence, all relevant techniques in the fields of operations research and combinatorial optimization have been tried on these problems, leading to a respected competition for the best results. Furthermore, the ability to solve many non-trivial problem instances exactly helps us to design or evaluate heuristics and relaxations. Additionally, to get a successful exact algorithm we have to develop extremely effective and efficient techniques for computing bounds and reducing the instances, so afterwards each (combination) of these components can be used, as adequate to the requirements. Finally, we even profit from the exact algorithms as a part of each of these components [35].

Due to the abundance of material, this article offers neither a comprehensive survey on the Steiner problem nor a detailed presentation of all our methods and results. Instead, we first provide in Section 2 a brief overview on the main algorithmic developments related to our work on this problem, citing our and others' (already published) works. Then we focus on some central concepts for dealing with ($\mathcal{NP}$-)hard optimization problems, namely relaxation and approximation. Thereby, we present in more detail (including some instructive or unpublished proofs) selected results that offer special insight and potential for further improvement. In Section 3 we present and relate the classical and our new relaxations, leading to a hierarchy of relaxations. In Section 4, we use full Steiner trees to provide some new insights into the integrality gap of the relaxations. We conclude in Section 5 with some remarks for further research.

1.1 Basic Definitions and Notations

The **Steiner problem in networks** can be stated as follows [16]: Given a weighted graph (or network) $G = (V, E, c)$ with vertices V, edges E, cost function $c : E \to \mathbb{R}$, and a set R, $\emptyset \neq R \subseteq V$, of **required vertices** (or **terminals**), find a minimum-cost subgraph of G that spans R (i.e., contains a path between every pair of terminals). Without loss of generality, we assume that the edge costs are positive and G is connected. Now there is an optimum solution that is a tree, called a **Steiner minimum tree (SMT).**

A **Steiner tree** is an acyclic, connected subgraph of G, spanning (a superset of) R. We call non-terminals in a Steiner tree its **Steiner nodes**. For better distinction, we sometimes denote terminals by z_i (represented by filled circles) and non-terminals by s_i (represented by empty circles). We denote by (v_i, v_j) the (undirected) edge $\{v_i, v_j\}$ and by $[v_i, v_j]$ the directed edge, or arc, from v_i to v_j. We denote the cost of each edge (v_i, v_j) (or arc $[v_i, v_j]$) by c_{ij}. For any network G, $c(G)$ denotes the sum of the edge costs of G.

For a given network G, the corresponding **distance network** is the complete network over the same vertices where the cost of each edge is the distance of its endpoints in G. It is easy to see that the optimum value of a Steiner tree (and of all relaxations considered here) is identical in G and its distance network. Therefore, we can work with the distance network (which is a metric instance) whenever needed (as in Section 4).

2 Brief Overview on the Main Algorithmic Developments

In this section we provide a brief overview on the major algorithmic developments in the context of our work on the Steiner problem in networks, thereby also listing our main contributions in this field. This section is structured according to the main components of a state-of-the-art program packet for dealing with this problem, namely lower bounds (Subsection 2.1), upper bounds (Subsection 2.3), reduction methods (Subsection 2.2), and exact algorithms (Subsection 2.4).

2.1 Relaxations and Lower Bounds

To attack a (hard) combinatorial optimization problem, a starting point could be a reformulation of the problem as an integer program. Dropping (relaxing) some of the constraints in such a program can make it more tractable. The solution of such a relaxation provides us with a lower bound (in case of minimization) for the original problem, which is a major component of many exact algorithms. But the information we get from handling such a relaxation can also be used beyond the computation of lower bounds, for example for reducing the instances and even for computing (heuristic) solutions (here: Steiner trees), see [35].

Some of the most successful approaches to hard combinatorial optimization problems are based on linear programming, consider as a celebrated example the case of the traveling salesman problem [3]. Here, the problem is first formulated as an integer linear program. Dropping the integrality constraints leads to a so-called **LP relaxation** (or **linear relaxation**) of the problem. In this way one can profit from the numerous sophisticated techniques for solving or approximating linear programs.

For $\mathcal{NP}$-hard optimization problems, unless $\mathcal{P} = \mathcal{NP}$, any linear relaxation of polynomial size (and any polynomial-time solvable relaxation) is bound to have a deviation from the solution of the original problem in some cases. The quality of the used relaxation can have a decisive impact on the performance of the algorithms based on it.

Below, we list our main contributions regarding the study and application of relaxations, thereby also referring to already published works.

- There have been many (mixed) integer programming formulations of the Steiner problem, but not much was known about the relative quality of their LP relaxations. We compared all classical, frequently cited relaxations from a theoretical point of view with respect to their optimal values. We could present several new results, establishing very clear relations between relaxations which have often been treated as unrelated or incomparable [25], and construct a hierarchy of relaxations (see Section 3.3).
- We introduced a collection of new relaxations that are stronger than all known relaxations that can be solved in polynomial time, and placed them into our hierarchy (see Section 3). Further, we derived a practical version of such a relaxation and showed how it can optimized efficiently [35].

- For geometric Steiner problems, the state-of-the-art algorithms follow a two-phase approach by first calculating certain full Steiner trees (see Section 4) and then solving a minimum spanning tree problem in the resulting hypergraph (MSTH) [38]. We studied the known and some new relaxations for the MSTH problem and showed the relationships between them and to the relaxations of the Steiner problem [28]. In this way, we also gain some new insights into the integrality gap of the relaxations (see Section 4).
- From an algorithmic point of view, the usefulness of a relaxation is decided not only by its optimum value, but also by the consideration how fast this value can be determined or sufficiently approximated. We analyzed and improved some algorithmic approaches to the relaxations [24,26,35].
- Especially in the context of exact algorithms, a major problem arises when none of the (practically tractable) relaxations is strong enough to solve the instance without branching. We presented two theoretically interesting and practically applicable techniques for improving the relaxations, namely graph transformation and local cuts. These techniques have proved to be quite powerful, especially for the solution of large and complex instances. In particular, the method of graph transformation, which was applied by us for the first time to a network optimization problem, seems quite promising [1].

2.2 Reductions to Simplify Problem Instances

Informally, reductions are here methods to reduce the size of a given instance without destroying the optimal solution. It has been known for some time that reductions can play an important role as a preprocessing step for the solution of $\mathcal{NP}$-hard problems. In particular, the importance of reductions for the Steiner problem has been widely recognized and a large number of techniques were developed [10,16]; a milestone was the PhD thesis of Cees Duin [11].

Since each reduction method is specially effective on a certain type of instances, and has less (or even no) effect on some others, it is important to have a large arsenal of different methods at one's disposal. The design and application of reduction techniques has been a central part of our work on the Steiner problem, with the main contributions listed below.

- For some of the classical reduction tests, which would have been too time-consuming for large instances in their original form, we could design efficient realizations, improving the worst-case running time to $O(|E| + |V| \log |V|)$ in many cases. Furthermore, we have designed new tests, filling some of the gaps left by the classical tests [26].
- Previous reduction tests were either alternative based or bound based. That means to simplify the instance they either argued with the existence of alternative solutions, or they used some constrained lower bound and upper bound. We developed a framework for extended reduction tests, which extends the scope of inspection of reduction tests to larger patterns and combines for the first time alternative-based and bound-based approaches [27].

- In the solution process, particularly as the result of our other reduction techniques, we frequently encounter graphs of (locally) low connectivity; but the standard methods based on partitioning are not helpful for exploiting this situation. We presented the new approach of using partitioning to design reduction methods. The resulting methods have been quite effective for the Steiner problem, and the approach can also be useful for other problems [29].
- We integrated all tests into a reduction packet, which performs stronger reductions than any other package we are aware of. Additionally, the reduction results of other packages can be achieved typically in a fraction of the running time (see [26,35] and also the comparison in [8]).

2.3 Heuristics and Upper Bounds

Since we cannot expect to solve all instances of an $\mathcal{NP}$-hard problem like the Steiner problem in times that are small (polynomial) with respect to the size of the given instance, methods for computing "good", but not necessarily optimal solutions (heuristics) are well motivated on their own. But the value of such a solution provides also an upper bound for the value of any optimal solution, which can be used, for example, in bound-based reductions or in exact algorithms.

For the Steiner problem, the number of papers on heuristics is enormous; there are many tailored heuristics, and also every popular meta-heuristic has been tried (often in many variants). Comprehensive overviews of older articles are given in [16,23], some more recent results can be found in [22,35].

We have developed a variety of heuristics for obtaining upper bounds. Especially in the context of exact algorithms, very sharp upper bounds are highly desired, preferably without resorting to very long runs.

- We developed variants of known path heuristics, including an empirically fast variant with worst-case running time $O(|E| + |V| \log |V|)$ [26].
- We introduced the new concept of reduction-based heuristics. On the basis of this concept, we developed heuristics that achieve in most cases sharper upper bounds than the strongest known heuristics for this problem despite running times that are smaller by orders of magnitude [26,35].

2.4 Exact Algorithms

For an $\mathcal{NP}$-hard problem like the Steiner problem, it is usually not considered as surprising that (not very large) instances can be constructed that defy existing exact algorithms. On the other hand, it should not be surprising that relatively large (non-trivial) instances can be solved exactly in fairly small times. An instructive case is the development in the context of the Steiner problem in recent years: Instances (of a size or type) that were assumed to be completely out of the reach of exact algorithms some years ago can now be solved in reasonable times. But this does not happen by itself, say due to faster computers: The improvements in the running times are often by orders of magnitude larger than the gains in the speed of the used computers. Major improvements happen when somebody comes up with new ideas that work well in practice.

For the Steiner problem, there are many published works on exact solution building upon every classical approach (enumeration algorithms, dynamic programming, branch-and-bound, branch-and-cut). Again, we point to [16] for a first survey. Later major contributions were presented by Duin [11] (advanced reductions and heuristics in a branch-and-bound framework), Koch and Martin [18] (improved branch-and-cut), and Uchoa et al. [34] (further improved reductions, again with branch-and-cut). For recent improvements of the worst-case time bounds of (dynamic programming) exact algorithms see [4,12].

Our major contribution in this context has been the integration of the components described in previous subsections into an "orchestra", so that we can largely profit from the resulting synergy effects.

- We presented an algorithm that exploits small width in (sub-)graphs, and showed how it can be used profitably in combination with our other techniques in a more general context [29].
- We showed how one can take advantage of the interaction between the components to design a very powerful reduction process. We integrated this reduction process into a branch-and-bound framework [26,35].

We have performed detailed experimental studies for each of the components and the whole exact solution program, thereby using the well-established benchmark library SteinLib [19,33].

- We could solve all instances in SteinLib that have been solved before, in most cases in running times that are smaller than those in the literature by orders of magnitude (see [26,35] and also the comparison in [8]).
- We have been able to solve many previously unsolved instances (from SteinLib). All still unsolved instances in SteinLib have been constructed to be difficult for known techniques.
- Regarding geometric Steiner problems, our algorithm for general networks has turned out to be competitive with a specially tailored MSTH approach [38], for large instances even outperforming it by orders of magnitude [35].

3 Relaxations for the Steiner Problem in Networks

As we have described before (Section 2.1), a deviation between the (optimal) solutions of a relaxation and the original integer problem can seriously impair the performance of the algorithms based on that relaxation. To counter this, two different approaches are possible. First, one can try to improve the relaxation without the explicit knowledge of a stronger relaxation (in advance); this is the approach we chose in [1], where we described two new and successful methods along this path. The other (more theoretical) approach is to come up with explicit alternative formulations that lead to stronger relaxations, which is the subject of this section.

For the following, we need some further notations. We use the notation P_Q for (mixed) integer linear programs, where the abbreviation Q describes the program further. The corresponding linear relaxation is denoted by LP_Q and the dual of such a relaxation is denoted by DLP_Q. These notations denote programs corresponding to an arbitrary, but fixed instance of the problem. The value of an optimal solution of a (mixed) integer programming formulation, denoted by $v(P_Q)$, is of course the value of an optimal solution of the corresponding (Steiner) problem; but the optimal solution value $v(LP_Q)$ of the corresponding linear relaxation can differ from $v(P_Q)$ (**integrality gap**).

We compare relaxations using the predicates **equivalent** and (**strictly**) **stronger**: We call a relaxation R_1 stronger than a relaxation R_2 if the optimal value of R_1 is no less than that of R_2 for all instances of the problem. If the converse is also true, we call them equivalent; otherwise we say that R_1 is strictly stronger than R_2. If neither is stronger than the other, they are **incomparable**.

We will briefly describe in Section 3.1 the classical cut relaxation and some variants and properties. In Section 3.2, we take a closer look at the integrality gap of the cut relaxation. This leads to the concept of common flows, which we use as the basis for developing stronger relaxations. We show how to derive various concrete relaxations from the generic template based on common flows, which turn out to be the strongest polynomial (i.e., of polynomial size or computable in polynomial time) relaxations known for the Steiner problem. We compare these new and all previous relaxations of this problem, leading to a new hierarchy of relaxations presented in Section 3.3.

3.1 The Cut and Flow Formulations

In this section, we briefly state the classical cut formulation, the equivalent multicommodity flow formulation, and some variants, properties and notations which are needed in the rest of the paper.

In the context of relaxations, sometimes it is advantageous to use a reformulation of the problem based on the (bi-)directed version of the given graph: Given $G = (V, E, c)$ and R, find a minimum-cost arborescence (directed tree) in $\boldsymbol{G} = (V, A, c)$ ($A := \{[v_i, v_j], [v_j, v_i] \mid (v_i, v_j) \in E\}$, c defined accordingly) with a terminal (say z_r) as the root that spans $R^{z_r} := R \backslash \{z_r\}$ (i.e., contains a path from z_r to every terminal in R^{z_r}).

A **cut** in a graph is defined here as a partition $(\overline{W}, W)$ of its vertex set V ($\emptyset \subset W \subset V; V = W \dot{\cup} \overline{W}$). We use $\delta^-(W)$ to denote the set of arcs $[v_i, v_j]$ with $v_i \in \overline{W}$ and $v_j \in W$. The sets $\delta^+(W)$ and, for the undirected version, $\delta(W)$ are defined similarly. For simplicity, we write $\delta^-(v_i)$ instead of $\delta^-(\{v_i\})$. A cut $C = (\overline{W}, W)$ is called a **Steiner cut** if $z_r \in \overline{W}$ and $R \cap W \neq \emptyset$.

In the (integer) linear programming formulations we use (binary) variables x_{ij} for each arc $[v_i, v_j] \in A$ (or X_{ij} for each edge $(v_i, v_j) \in E$), indicating whether an arc is part of the solution ($x_{ij} = 1$) or not ($x_{ij} = 0$). Thus, the cost of the solution can be calculated by the dot product $c \cdot x$, where c is the cost vector. For any $B \subseteq A$, $x(B)$ is short for $\sum_{b \in B} x_b$; for example, $x(\delta^-(W))$ is short for $\sum_{[v_i,v_j] \in A, v_i \notin W, v_j \in W} x_{ij}$.

Directed Cut Formulation. The directed cut formulation was stated in [39]. An undirected version was already introduced in [2], but the directed variant yields a strictly stronger relaxation [7].

$$\boxed{P_C} \qquad c \cdot x \rightarrow \min,$$

$$x(\delta^-(W)) \geq 1 \qquad (z_r \notin W; R \cap W \neq \emptyset), \tag{1.1}$$

$$x \in \{0,1\}^{|A|}. \tag{1.2}$$

The constraints (1.1) are called **Steiner cut constraints**. They guarantee that in any arc set corresponding to a feasible solution, there is a path from z_r to any other terminal.

Multicommodity Flow Formulation. Viewing the Steiner problem as a multicommodity flow problem leads to the following formulation [39].

$$\boxed{P_F} \qquad c \cdot x \rightarrow \min,$$

$$y^t(\delta^-(v_i)) = y^t(\delta^+(v_i)) + \begin{cases} 1 & (z_t \in R^{z_r}; v_i = z_t), \\ 0 & (z_t \in R^{z_r}; v_i \in V \setminus \{z_r, z_t\}), \end{cases} \tag{2.1}$$

$$y^t \leq x \qquad (z_t \in R^{z_r}), \tag{2.2}$$

$$y^t \geq 0 \qquad (z_t \in R^{z_r}), \tag{2.3}$$

$$x \in \{0,1\}^{|A|}. \tag{2.4}$$

Each variable y^t_{ij} denotes the quantity of the commodity t flowing through $[v_i, v_j]$. Constraints (2.1) and (2.3) guarantee that for each terminal $z_t \in R^{z_r}$, there is a flow of one unit of commodity t from z_r to z_t. Together with (2.2), they guarantee that in any arc set corresponding to a feasible solution, there is a path from z_r to any other terminal.

Lemma 1. *LP_F is equivalent to LP_C. In fact, using the max-flow min-cut theorem, it is easy to see that every feasible solution $\hat{x}$ for LP_C corresponds to a feasible solution $(\hat{x}, \hat{y})$ for LP_F and vice versa [16].*

An Augmented Flow Formulation. Observing that in an optimal (directed) Steiner tree, the out-degree of each Steiner node is not smaller than its in-degree, one can use the following constraints (which we call flow-balance (FB) constraints) to make LP_F stronger [11,18].

$$x(\delta^-(v_i)) \leq x(\delta^+(v_i)) \qquad (v_i \in V \setminus R) \tag{3.1}$$

We denote the linear program that consists of LP_F (or LP_C) and (3.1) by LP_{F+FB} (or LP_{C+FB}). In [25] we showed that LP_{F+FB} is strictly stronger than LP_F. Now consider the following formulation.

$$\boxed{P_{F'+FB}} \qquad c \cdot X \rightarrow \min,$$

$$x_{ij} + x_{ji} = X_{ij} \qquad ((v_i, v_j) \in E), \tag{4.1}$$

$$(x, y) \ : \ \text{is feasible for } P_{F+FB}. \tag{4.2}$$

Lemma 2. *If (X, x, y) is a solution for $LP_{F'+FB}$ with root z_a, then there exists a solution $(X, \check{x}, \check{y})$ for $LP_{F'+FB}$ with any other root $z_b \in R\backslash\{z_a\}$.*

Proof. It is easy to verify that $(X, \check{x}, \check{y})$ with $\check{x}_{ij} := x_{ij} + y^b_{ji} - y^b_{ij}$, $\check{y}^t_{ij} := \max\{0, y^t_{ij} - y^b_{ij}\} + \max\{0, y^b_{ji} - y^t_{ji}\}$, $\check{y}^a_{ij} := y^b_{ji}$ (for $[v_i, v_j] \in A$, $z_t \in R\backslash\{z_a, z_b\}$) satisfies (4.1) and (2.2). Because of $\sum_{[v_j,v_i]\in\delta^-(v_i)}(\check{y}^t_{ji} - \check{y}^t_{ij}) = \sum_{[v_j,v_i]\in\delta^-(v_i)}$ $(\max\{0, y^t_{ji} - y^b_{ji}\} + \max\{0, y^b_{ij} - y^t_{ij}\} + \min\{0, -y^t_{ij} + y^b_{ij}\} + \min\{0, -y^b_{ji} + y^t_{ji}\}) = \sum_{[v_j,v_i]\in\delta^-(v_i)}(y^t_{ji} - y^t_{ij}) - (y^b_{ji} - y^b_{ij})$ (for $v_i \in V$, $z_t \in R\backslash\{z_a, z_b\}$) the constraints (2.1) are satisfied, too. From (2.1) for y^b follows that $x(\delta^-(v_i)) = \check{x}(\delta^-(v_i))$ and $x(\delta^+(v_i)) = \check{x}(\delta^+(v_i))$ (for $v_i \in V\backslash R$), so $\check{x}$ satisfies the constraints (3.1). □

Because this translation could also be performed from any (optimal) solution with root z_b to a feasible solution with root z_a, the value $v(LP_{F'+FB})$ (or $v(LP_{F+FB})$) is independent of the choice of the root; and $LP_{F'+FB}$ is equivalent to LP_{F+FB} (and LP_{C+FB}).

3.2 A Collection of New Formulations

In this section, we demonstrate how the study of the weaknesses of known relaxations can lead to the development of stronger relaxations.

Integrality Gap of the Flow/Cut Relaxations. An instance with a deviation between the integral and the linear solution for the (directed) flow formulation is described in the following example.

Example 1. For the instance shown in Figure 1 (say with z_1 as the root), we have $v(LP_F) = 7\frac{1}{2}$ (setting x-variables to 1/2 for the arcs $[z_1, s_2]$,$[z_1, s_3]$, $[s_1, z_2]$,$[s_3, z_2]$, $[s_1, z_3]$,$[s_2, z_3]$, $[s_2, v_0]$,$[s_3, v_0]$,$[v_0, s_1]$); while an optimal Steiner tree has the cost 8. Thus the integrality gap (as the ratio $v(P_F)/v(LP_F)$) is at least 16/15.

Goemans [13] extended this example to graphs G_k with $k+1$ terminals (Figure 1 shows G_2) whose integrality gap comes arbitrarily close to 8/7 with growing k; and no example with $v(P_F)/v(LP_F)$ larger than 8/7 is known [35]. However, no better upper bound than 2 for this ratio has been proved in general [14,24].

Looking at the flow-variables y^2 and y^3 in Example 1, one can see that at the vertex v_0 flows of two commodities enter over different arcs ($[s_2, v_0]$ and $[s_3, v_0]$), but depart together over the arc $[v_0, s_1]$. We denote this situation as a rejoining. Formally, a **rejoining** of flows of the commodities for $\{z_{i_1}, z_{i_2}, \ldots, z_{i_p}\} =: B$ at a vertex v_i is defined by the condition

$$\sum_{[v_j,v_i]\in\delta^-(v_i)} \max\{y^t_{ji} | z_t \in B\} > \sum_{[v_i,v_k]\in\delta^+(v_i)} \max\{y^t_{ik} | z_t \in B\}.$$

Later we will show how to counter this problem in an extended linear program.

This situation is typical. In fact, we do not know of any graph with an integrality gap for LP_F that does not include a variant of Figure 1. Of course,

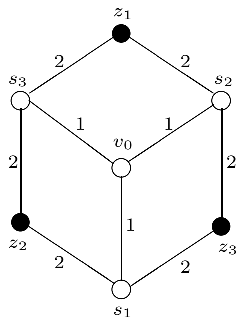


Fig. 1. Instance with $v(LP_F) < v(P_F)$

there may be additional edges, edges replaced by paths, other edge costs and non-terminals replaced by terminals, but still the graph in Figure 1 is a minor of any graph with an integrality gap (a minor of a graph is obtained by a sequence of deletions and contractions, for the relevance of minors to linear relaxations of the Steiner problem see [7]). Furthermore, the rejoining of flows of different commodities is a common property of all examples with an integrality gap known to us, although more than two flows can rejoin (as we will show in Figure 2) and it may need a change of the root to see a rejoining. Figure 1 can be used to illustrate the latter: If we turn v_0 into a terminal z_0 and choose it as root, the optimal linear solution value stays the same, but no rejoining of flows shows up.

Common Flow. As we have seen above, the basic problematic situation for LP_F can be described as the rejoining of flows of different commodities. We capture this in linear constraints with the help of additional variables y^B (for subsets $B \subseteq R^{z_r}$) that describe the amount of flow going from the root to any terminal $z_t \in B$: $y^B \geq y^t$ for all $z_t \in B$. In Figure 1, it is easy to see that the flow to z_2 and z_3 ($y^{\{z_2,z_3\}}$) incoming at v_0 is greater than the outgoing flow. This cannot be the case for x-values corresponding to a feasible Steiner tree. Thus, we introduce additional constraints to ensure that for each common flow the outgoing amount at any vertex is not smaller than the incoming amount.

As already noted, the same graph can or cannot show rejoining of flows, depending on the choice of the root. To detect all rejoinings we consider all possible roots. Thus, the corresponding variables in the common-flow formulation will have the shape $y^{r,B}$, describing a flow from z_r to all terminals $z_t \in B \subseteq R^{z_r}$.

A Template for Common-Flow Formulations. We first state an exhaustive common-flow formulation that has an exponential number of constraints and variables, and describe afterwards how to reduce it to different formulations (e.g., of only polynomial size).

$$\boxed{P_{F^R}} \qquad c \cdot X \to \min,$$

$$
\begin{aligned}
y^{r,\{z_t\}}(\delta^-(z_r)) &= y^{r,\{z_t\}}(\delta^+(z_r)) - 1 && (z_r \in R; z_t \in R^{z_r}), && (5.1)\\
y^{r,\{z_t\}} - y^{r,\{z_s\}} &\le y^{s,\{z_t\}} && (\{z_r, z_s, z_t\} \subseteq R), && (5.2)\\
y_{ji}^{r,\{z_s\}} - y_{ji}^{r,\{z_t\}} &\le y_{ij}^{s,\{z_t\}} && (\{z_r, z_s, z_t\} \subseteq R; [v_i, v_j] \in A), && (5.3)\\
y^{r,B}(\delta^-(v_i)) &\le y^{r,B}(\delta^+(v_i)) && (z_r \in R; B \subseteq R^{z_r}; v_i \in V \setminus (B \cup \{z_r\}), && (5.4)\\
y^{r,B} &\le y^{r,C} && (z_r \in R; B \subset C \subseteq R^{z_r}), && (5.5)\\
y_{ij}^{r,R^{z_r}} + y_{ji}^{r,R^{z_r}} &\le X_{ij} && (z_r \in R; (v_i, v_j) \in E), && (5.6)\\
y^{r,\{z_r\}} &= 0 && (z_r \in R), && (5.7)\\
y &\ge 0, && && (5.8)\\
X &\in \{0,1\}^{|E|}. && && (5.9)
\end{aligned}
$$

It follows from the proof of Lemma 2 that the constraints (5.2) and (5.3) redirect the flows from a root z_r to flows from any other root $z_s \in R$ to each terminal $z_t \in R$. In particular, from these constraints in combination with (5.7) it follows that $y_{ji}^{t,\{z_r\}} = y_{ij}^{r,\{z_t\}}$ for all $z_r, z_t \in R$. For each flow from a root z_r to a terminal z_t, described by the variables $y^{r,\{z_t\}}$, the constraints (5.1) guarantee an outflow of one unit out of z_r, and (together with the previous observation) an inflow of one unit in z_t. Together with the constraints (5.4) (with $B = \{z_t\}$) and (5.8) this guarantees a flow of one unit from z_r to each terminal $z_t \in R^{z_r}$. The constraints (5.4) also guarantee for any root z_r and set of terminals B that there is no rejoining at any vertex v_i (to be more precise, there is a penalty for rejoined flows in form of increased $y^{r,B}$ values). The constraints (5.5) set each common flow from a root z_r to a set of terminals C to at least the maximum of all flows from z_r to a subset $B \subset C$ of the terminals. Finally the constraints (5.6) build the edgewise maximum over all flows from all terminals in the edge variables X.

We do not know of any graph where the relaxation LP_{F^R} has an integrality gap. Note that due to the exponential number of constraints *and* variables the possibility that there is no integrality gap is not ruled out by known complexity arguments (and common assumptions).

Polynomial Variants. To derive polynomial-size relaxations from LP_{F^R}, we limit the number of terminal sets B polynomially, e.g., by choosing two constants $k_1 \ge 1$ and $k_2 \ge 1$ ($k_1 + k_2 < |R|$), and using only those variables $y^{r,B}$ with $|B|$ either at most k_1 or at least $|R| - k_2$. We denote the resulting relaxation by $LP_{F^{k_1,k_2}}$. It has $O(|R|^{1+\max\{k_1,k_2-1\}}|A|)$ variables and $O(|R|^{2\max\{k_1,k_2-1\}}|A|)$ constraints. For example, if we choose $k_1 = 2$ and $k_2 = 1$ and also use only one fixed root z_r we get a relaxation equivalent to LP_{F^2} which we presented in [25]. Here, we state that relaxation in the form as it is derives from P_{F^R}, using the variables $y^{r,B}$ for a fixed z_r and for $B \in \mathcal{B} := \{C \subseteq R^{z_r} \mid |C| \in \{1, 2, |R| - 1\}\}$.

$$\begin{aligned}
\boxed{P_{F^2}} \qquad & c \cdot x \to \min, \\
y^{r,\{z_t\}}(\delta^-(z_r)) &= y^{r,\{z_t\}}(\delta^+(z_r)) - 1 \quad (z_t \in R^{z_r}), && (6.1)\\
y^{r,B}(\delta^-(v_i)) &\le y^{r,B}(\delta^+(v_i)) \qquad (B \in \mathcal{B}; v_i \in V \setminus (B \cup \{z_r\}), && (6.2)\\
y^{r,B} &\le y^{r,C} \qquad (B \subset C \in \mathcal{B}), && (6.3)\\
y^{r,R^{z_r}} &= x, && (6.4)\\
y &\ge 0, && (6.5)\\
x &\in \{0,1\}^{|A|}. && (6.6)
\end{aligned}$$

The drawback of this limitation is that not all rejoinings will be detected by the limited relaxation, as the following example shows.

Example 2. In Figure 2, using LP_F with z_1 as the root, there is a rejoining of 3 different flows (y^2, y^3 and y^4) at v_0 (an optimal solution is obtained by setting the x-variables to 1/3 for all (existing) arcs $[z_1, s_i]$,$[s_i, z_2]$,$[s_i, z_3]$,$[s_i, z_4]$, $[s_2, v_0]$,$[s_3, v_0]$,$[s_4, v_0]$,$[v_0, s_1]$ and to 1 for the arc $[v_0, z_0]$). As LP_{F^2} captures only the rejoining of 2 flows, there will still be some integrality gap.

The figure also shows how the choice of k_2 can affect the relaxation: Only if the common flow $y^{r,B}$ to a set of terminals B is considered such that $z_2, z_3, z_4 \in B$, but $z_0 \notin B$, the relaxation gives the optimal integer value. On the other hand, for $k_1 \ge 3$ or $k_2 \ge 2$ there is no integrality gap.

Such examples can be generalized to graphs $G[k_1, k_2]$ that produce an integrality gap for any common-flow relaxation limited by some constants k_1 and k_2 (Figures 1 and 2 show the graphs $G[1, 0]$ and $G[2, 1]$): Around a vertex v_0 put $k_1 + 2$ non-terminals s_i and connect them to v_0 with edges of cost 1. Add $k_1 + 2$ terminals z_j and connect each z_j to all s_i, $j \ne i$, with edges of cost $k_1 + 1$. Finally, connect v_0 to k_2 new terminals z_0^l with edges of cost 1. An integer optimal solution has the cost $k_2 + (k_1 + 1)(k_1 + 2) + 2$. On the other hand, a feasible solution for the (k_1, k_2)-limited common-flow relaxation can be constructed as follows: Set the X-variables for edges incident to $z_0^l, 1 \le l \le k_2$, to 1; the X-variable for (v_0, s_1) to $1 - (k_1 + 1)^{-1}$ and all other X-variables to $(k_1 + 1)^{-1}$. Thus, the optimal value of the relaxation is at most $k_2 + (k_1 + 1)(k_1 + 2) + 2 - (k_1 + 1)^{-1}$. As a consequence, we have derived a collection of relaxations $LP_{F^{k_1,k_2}}$ such that $LP_{F^{k_1,k_2}}$ can be related to $LP_{F^{j_1,j_2}}$; we assume that $k_1 + k_2 \le j_1 + j_2$:

if $k_1 \le j_1, k_2 \le j_2$**:** The relaxation $LP_{F^{j_1,j_2}}$ is stronger than $LP_{F^{k_1,k_2}}$ (since it contains a superset of the constraints and variables). If additionally $k_1 < j_1$ or $k_2 < j_2$, the relaxation $LP_{F^{j_1,j_2}}$ is even strictly stronger. Consider the graph $G[\min\{k_1, j_1\}, \min\{k_2, j_2\}]$: $LP_{F^{j_1,j_2}}$ gives the optimal integral solution, while $LP_{F^{k_1,k_2}}$ has an integrality gap.

otherwise: The relaxations are incomparable, consider the two graphs $G[\min\{k_1, j_1\}, \max\{k_2, j_2\}]$ and $G[\max\{k_1, j_1\}, \min\{k_2, j_2\}]$.

Now, we show why considering all roots is important. The only difference between the relaxations $LP_{F^{2,1}}$ and LP_{F^2} is that $LP_{F^{2,1}}$ considers all terminals as root.

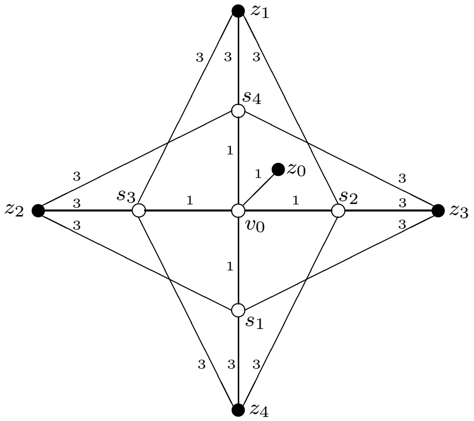


Fig. 2. $v(LP_F) = 14\frac{1}{3} < v(LP_{F^{2,1}}) = 14\frac{2}{3} < v(LP_{F^R}) = 15 = v(P_{F^R})$

Lemma 3. *The relaxation $LP_{F^{2,1}}$ is strictly stronger than LP_{F^2}.*

Proof. The relaxation $LP_{F^{2,1}}$ is stronger than LP_{F^2}, as it contains a superset of the constraints and variables. To see that it is even strictly stronger, consider the graph $G[1,1]$ (Figure 1 with an extra edge (v_0, z_0)). In this graph, $LP_{F^{2,1}}$ has no integrality gap, but for LP_{F^2} it depends on the choice of the root: With z_1, z_2 or z_3 - but not with z_0 - as the root we get the optimal integer value. □

Practical Adaptations. Here we present - as an example - a quite restricted, but still very useful variant of LP_{F^R}.

We observed that in many cases the tracing of the common flow to all $|R| - 1$ terminals ($k_2 = 1$) catches most of the problematic situations. As depicted in Figure 2, to fool the relaxation with $k_2 = 1$, typically there has to be a terminal directly connected to a vertex at which a rejoining happens. In many such cases, the terminal is the vertex itself. To counter such a situation without introducing a quadratic number of constraints or variables, we restrict the common-flow relaxation by $k_1 = k_2 = 1$ on non-terminals, and by $k_2 = 2$ on terminals, but only considering the set $R \setminus \{z_r, z_i\}$ for any terminal z_i (with respect to root z_r). Additionally, we restrict the relaxation to one fixed root z_r, and because we do not need the X-variables anymore, we can replace the $y^{R^{z_r}}$ with x-variables.

In a cut-and-price framework it is highly desirable to introduce as few variables as possible. Working with flow-based relaxations makes this difficult, because if a flow variable for one edge is needed, all variables and all flow conservation constraints have to be inserted to get something useful. Therefore, the cut-based formulations are more suitable in this context. In analogy to P_C, we can use constraints of the form $y^{\{z_i\}}(\delta^-(W)) \geq 1$ for all $W \cap \{z_r, z_i\} = \{z_i\}$. Finally, we eliminate the variables $y^{\{z_i\}}$ and $y_e^{R \setminus \{z_r, z_i\}}$ with $e \notin \delta^-(z_i)$: The $y^{\{z_i\}}$-variables are replaced by $y^{R \setminus \{z_r, z_i\}}$ if the replacement is possible using the constraints (5.5); and the $y^{R \setminus \{z_r, z_i\}}$-variables are replaced by x if possible.

All constraints that still use unwanted variables are deleted. Now, we have the following formulation:

$$\begin{aligned}
\boxed{P_{C'}} \quad & c \cdot x \rightarrow \min, && \\
x(\delta^-(W)) &\geq 1 && (z_r \notin W, R \cap W \neq \emptyset), && (7.1)\\
y^{R\setminus\{z_r,z_i\}}(\delta^-(W) \cap \delta^-(z_i)) + & && && \\
x(\delta^-(W) \setminus \delta^-(z_i)) &\geq 1 && (z_r \notin W, R^{z_i} \cap W \neq \emptyset), && (7.2)\\
x(\delta^-(v_i)) &\leq x(\delta^+(v_i)) && (v_i \in V \setminus R), && (7.3)\\
y^{R\setminus\{z_r,z_i\}}(\delta^-(z_i)) &\leq x(\delta^+(z_i)) && (z_i \in R^{z_r}), && (7.4)\\
y &\geq 0, && && (7.5)\\
x &\in \{0,1\}^{|A|}. && && (7.6)
\end{aligned}$$

This new formulation has less than $2|A|$ variables. Although it has an exponential number of constraints, its linear relaxation can be solved efficiently by row and column generation, see [35].

Relation to Other Relaxations

Lemma 4. *The relaxation $LP_{F^{1,1}}$ is equivalent to $LP_{F'+FB}$.*

Proof. The relaxations are very similar. An apparent difference is that $LP_{F^{1,1}}$ considers all possible roots. But as we have shown in the proof of Lemma 2, the choice of the root does not change the value of a solution for $LP_{F'+FB}$; and the constraints (5.2) and (5.3) in $LP_{F^{1,1}}$ cover exactly the transformation of a solution for one root into a solution for another. □

Lemma 5. *The relaxation LP_{F^2} is strictly stronger than $LP_{C'}$ and $LP_{C'}$ is strictly stronger than $LP_{F^{1,1}}$.*

Proof. Obviously LP_{F^2} is stronger than $LP_{C'}$, as $LP_{C'}$ is a restricted and aggregated version of LP_{F^2}. Similarly, $LP_{C'}$ contains a superset of the variables and constraints of LP_{C+FB}, which is equivalent to $LP_{F^{1,1}}$ (Lemma 4). To see that the relations are strict, consider the graph $G[1,1]$ (Figure 1 with an extra edge (v_0, z_0)). If we choose z_1, z_2 or z_3 as the root, $LP_{C'}$ has an integrality gap, but not LP_{F^2}; and if we contract (v_0, z_0), LP_{C+FB} still has a gap, but not $LP_{C'}$. □

3.3 A Hierarchy of Relaxations

Figure 3 summarizes the relations stated before. All relaxations in the same box are equivalent. A line between two boxes means that the relaxations in the upper box are strictly stronger than those in the lower box. Notice that the “strictly stronger” relation is transitive. All relaxations (abbreviations) not described here were already considered in [25]; all of them are weaker than the relaxations presented in this paper.

This hierarchy clarifies the relationship between all published relaxations for the Steiner problem and shows explicitly how new (arbitrarily strong) relaxations can be developed by introducing additional variables.

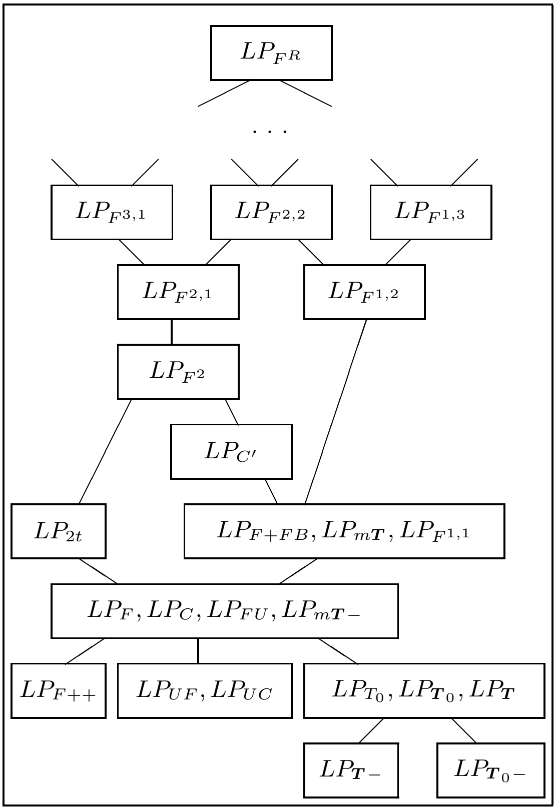


Fig. 3. Hierarchy of relaxations

4 Relaxation and Approximation by Full Steiner Trees

Many exact and approximative approaches to the Steiner problem use the concept of full Steiner trees. Any Steiner minimum tree (SMT) can be decomposed into its so-called full components by splitting its interior terminals; each resulting subtree is a full Steiner tree. (A **full Steiner tree (FST)** for $Q \subseteq R$ is a tree with Q as its sets of leaves and terminals.) An approach that suggests itself naturally is trying to construct a (minimum) Steiner tree by considering (minimum) FSTs for subsets of terminals.

As far as exact algorithms are concerned, the application of FSTs has been particularly successful in the context of geometric Steiner problems [38]. In geometric Steiner problems, a set of points (in the plane) is to be connected at minimum cost according to some geometric distance metric. The FST approach consists of two phases. In the first phase, the FST generation phase, a set of FSTs is generated that is guaranteed to contain an SMT. In the second phase, the FST concatenation phase, one chooses a subset of the generated FSTs whose concatenation yields an SMT. Although there are point sets that give rise to an exponential number of FSTs in the first phase, usually only a linear number of

FSTs are generated, and empirically the bottleneck of this approach has usually been the second phase, where originally methods like backtracking or dynamic programming have been used. A breakthrough occurred as Warme [37] observed that FST concatenation can be reduced to finding a minimum spanning tree in a hypergraph whose vertices are the terminals and whose hyperedges correspond to the generated FSTs. Although the minimum spanning tree in hypergraph (**MSTH**) problem is $\mathcal{NP}$-hard, a branch-and-cut approach based on the linear relaxation of an integer programming formulation of this problem has been empirically successful [38].

As far as approximation algorithms are concerned, all algorithms with a performance ratio better than 2 use the concept of FSTs (see [15] for an extensive coverage of themes concerning the approximation of Steiner trees). The performance ratio of 2 can already be achieved by computing a minimum spanning tree (**MST**) for terminals in the distance network (so using FSTs with two terminals as edges). Almost all these approximation algorithms use a greedy strategy to improve the current Steiner tree (starting for example with the above mentioned MST) by considering certain FSTs (for example, all FSTs with at most a constant number k of terminals). In each step, an FST is chosen which maximizes a certain function (for example, the reduction in the cost of the current Steiner tree by using the non-terminals of an FST), until no eligible FST leads to an improvement anymore. Among these algorithms, the loss-contracting algorithm (which will be described in Section 4.3) achieves the best performance ratio, namely 1.55 for general and 1.28 for **quasi-bipartite** graphs (graphs with no edges between non-terminals).

In this section, we use the concept of FSTs to provide some insights into the integrality gap of the cut relaxation. In particular, we prove an improved bound of 1.39 on this gap for the special class of quasi-bipartite graphs.[1] We begin with the used relaxations and their relevant properties in Section 4.1. In particular, we present another cut-based relaxation which uses variables for (directed) FSTs. In Section 4.2, we clarify the relation between this relaxation and the classical cut relaxation. This relation enables us to profit from the performance ratio of a certain approximation algorithm for bounding the integrality gap of the considered relaxations, as demonstrated in Section 4.3.

4.1 Minimum Spanning Trees in Hypergraphs: Formulations

Let F be a set of FSTs. By identifying each FST $T \in F$ with its set of terminals, we get a hypergraph $H = (R, F)$. For each FST T, let c_T be the sum of its edge costs. Any FST T can be rooted from each of its l leaves, leading to a set of directed FSTs $\{\boldsymbol{T}_1, \ldots, \boldsymbol{T}_l\}$. We denote the set of directed FSTs generated from F in this way by $\boldsymbol{F}$. In the following, we use the term FST both for the tree T and the corresponding hyperedge in H, the meaning should be clear from the context. Cuts in hypergraphs can be defined similarly to

[1] Very recently, Chakrabarty, Devanur and Vazirani showed a bound of 4/3 for this value, using a different approach [5].

those in usual graphs (Section 3.1); here we use Δ instead of δ (for example, $\Delta(W) := \{T \in F \mid T \cap W \neq \emptyset, T \cap \overline{W} \neq \emptyset\}$).

We begin with a packing formulation of the MSTH problem, which has a variable X_T for each hyperedge (FST) T.

$$\boxed{P_{FST}} \qquad \sum_{T \in F} c_T X_T \rightarrow \min,$$

$$\sum_{T \in F} (|T| - 1) X_T = |R| - 1, \tag{8.1}$$

$$\sum_{T,\ T \cap W \neq \emptyset} (|T \cap W| - 1) X_T \leq |W| - 1 \quad (\emptyset \neq W \subset R), \tag{8.2}$$

$$X_T \in \{0,1\} \qquad (T \in F). \tag{8.3}$$

In [37], the following result (with a slightly different syntax) was shown.

Lemma 6. *Any feasible solution of P_{FST} describes a spanning tree for the hypergraph (R, F) and vice versa.*

Now consider the following (directed) cut formulation of the MSTH problem:

$$\boxed{P_{FSC}} \qquad \sum_{\boldsymbol{T} \in \boldsymbol{F}} c_{\boldsymbol{T}} X_{\boldsymbol{T}} \rightarrow \min,$$

$$\sum_{\boldsymbol{T} \in \boldsymbol{F}} (|\boldsymbol{T}| - 1) X_{\boldsymbol{T}} = |R| - 1, \tag{9.1}$$

$$\sum_{\boldsymbol{T},\ \boldsymbol{T} \in \Delta^-(W)} X_{\boldsymbol{T}} \geq 1 \qquad (z_r \notin W,\ W \cap R \neq \emptyset), \tag{9.2}$$

$$X_{\boldsymbol{T}} \in \{0,1\} \quad (\boldsymbol{T} \in \boldsymbol{F}). \tag{9.3}$$

In [28], we have shown the following result:

Lemma 7. *LP_{FSC} is equivalent to LP_{FST}.*

The equivalence is actually stronger: The sets of feasible solutions (and corresponding polyhedra) are identical for both relaxations. With respect to optimal solutions, our assumption that the edge costs are positive leads directly to the observation that $\sum_{\boldsymbol{T} \in \Delta^-(z_r)} X_{\boldsymbol{T}} = 0$. A more detailed analysis (similar to our proofs of Lemmas 8 and 9 in [25] for the directed cut relaxation in graphs) leads to the observation that for any optimal solution for LP_{FSC} without (9.1) and for every $z_k \in R^{z_r}$, it holds that $\sum_{\boldsymbol{T} \in \Delta^-(z_k)} X_{\boldsymbol{T}} = 1$. So dropping the constraints (9.1) does not change the optimal solution value of LP_{FSC}. In the following, we assume that the constraints (9.1) are omitted. Now we get the following dual program of LP_{FSC}, which has a variable u_W (only) for each Steiner cut $(\overline{W}, W)$.

$$\boxed{DLP_{FSC}} \qquad \sum u_W \rightarrow \max,$$

$$\sum_{W,\ \boldsymbol{T} \in \Delta^-(W)} u_W \leq c_{\boldsymbol{T}} \quad (\boldsymbol{T} \in \boldsymbol{F}), \tag{10.1}$$

$$u \geq 0. \tag{10.2}$$

For later use we note that by replacing $\boldsymbol{F}$ with A, we get the dual DLP_C of the cut relaxation LP_C.

4.2 Relating the Relaxations in Hypergraphs and Graphs

Let F be a set of FSTs in a graph G with terminals R. By building the union of (the edge sets of) the FSTs in F, we get a subgraph G_F of G. We assume that G_F contains a Steiner tree for R.

Lemma 8. *Consider the relaxations LP_{FSC} for (R, F) and LP_C for (G_F, R). We have $v(LP_{FSC}) \geq v(LP_C)$.*

Proof. Let $\hat{X}$ be an optimal solution of LP_{FSC}. For each arc $[v_i, v_j] \in A$ that is part of directed FSTs $\boldsymbol{T}_1, \ldots, \boldsymbol{T}_q$, let $\hat{x}_{ij} := \hat{X}_{\boldsymbol{T}_1} + \cdots + \hat{X}_{\boldsymbol{T}_q}$. It is easy to verify that $\hat{x}$ is feasible for LP_C and yields the same value $v(LP_{FSC})$. □

Lemma 9. *Let G be a quasi-bipartite graph. Then there exists a set of FSTs F in G such that $v(LP_{FSC})$ for the hypergraph (R, F) is at most $v(LP_C)$ for the Steiner problem instance (G, R).*

Proof. Let s be a non-terminal and $Z_s \subseteq R$ the set of vertices adjacent to s. Let $\hat{x}$ be an optimal solution to LP_C for $(\boldsymbol{G}, R)$ and $(\hat{x}, \hat{y})$ a corresponding optimal solution for LP_F. Let z_k be an arbitrary terminal in Z_s. For any other terminal $z_{k'} \in R^{z_r}$ with a flow of commodity k' of value $\hat{y}_P^{k'}$ along a path $P = z_l \rightarrow s \rightarrow z_k$, it can be ensured that there exists also a flow $\hat{y}_P^k$ of at least the same quantity along P (property †). To see this, observe that because of the constraints (2.1) (for terminal z_k), the $\hat{y}^k$-values over paths not including P from z_r to z_k sum up to at least $1 - \hat{y}_P^k$, so P can be replaced (if necessary) by alternative subpaths (without enhancing any $\hat{x}$-value) such that the constraints (2.1) remain valid for any k' and the property † is fulfilled.

Now consider all positive flows along paths containing the arc $[z_l, s]$: The successors of s in these paths define a tree $\boldsymbol{T}_l^1$ with root z_l, Steiner node s and leaves from Z_s. Let $\hat{y}_P^k$ along a path $P = z_l \rightarrow s \rightarrow z_k$ be the smallest such flow over all $z_k \in Z_s - \{z_l\}$. Imagine we introduce a disjoint copy of $\boldsymbol{T}_l^1$ with all $\hat{x}$-values equal to $\hat{y}_P^k$ and reduce the original $\hat{x}$-values correspondingly. Property † guarantees that a solution of the same value exists in the (imaginary) modified graph. By repeating this operation until $\hat{x}_{[z_l,s]}$ (for all s) gets zero, we get a set of directed FSTs $\boldsymbol{T}_l^1, \ldots, \boldsymbol{T}_l^j$ and corresponding $\hat{X}$-values. By repeating this process for all terminals z_l, we get a set $\boldsymbol{F}$ of directed FSTs such that $\hat{x}_{ij} = \sum_{\boldsymbol{T} \in \boldsymbol{F}: [v_i, v_j] \in \boldsymbol{T}} \hat{X}_{\boldsymbol{T}}$. Now it is easy to verify that $\hat{X}$ satisfies all constraints of LP_{FSC} (without (9.1)) and yields the same value as $\hat{x}$ in LP_C. □

From Lemmas 8 and 9, we immediately get the following theorem.

Theorem 1. *For any quasi-bipartite graph G with terminals R and any set F that contains all (optimal) FSTs in G the relaxations LP_C (for (G, R)) and LP_{FSC} (for (R, F)) have the same optimum value.*

In the following, we treat LP_{FSC} as a relaxation for the Steiner problem: Given an instance (G, R) of the Steiner problem, we assume that F contains all (optimal) FSTs in G (for each possible subset of R). When writing LP_{FSC} for G we mean the corresponding LP for the instance (R, F) of the MSTH problem.

4.3 Consequences for the Integrality Gap of Relaxations

First, we need some further definitions (see [15,32] for more details). A Steiner tree that does not contain any Steiner nodes is called a **terminal-spanning tree**. Let T^* be a terminal-spanning tree and T an FST. The minimum-cost subgraph $T^*[T]$ in $T^* \cup T$ that contains T and spans all terminals is again a Steiner tree, and the **gain** of T with respect to T^* is $gain_{T^*}(T) := \max\{0, c(T) - c(T^*[T])\}$. Let T be an FST. The **loss** of T is the cost of the cheapest subgraph of G (a forest) that connects the Steiner nodes of T to its terminals.

The loss-contracting algorithm [32] begins with the distance network of G as G_0 and an MST for R in G_0 as the initial terminal-spanning tree T_0. In each iteration i, an FST T is chosen that maximizes the ratio $gain_{T_{i-1}}(T)/loss(T)$. Then, an edge set corresponding to the loss of T is contracted (their cost is set to zero) in the graph G_{i-1}, leading to the graph G_i and a corresponding MST T_i. This process is repeated until we have $gain_{T_{i-1}}(T) \leq 0$ for all FSTs T. Now, we have the final loss-contracted graph G^* and the final terminal-spanning tree T^*. The tree T^*, together with the contracted edges of total cost $loss^*$ represent a feasible solution (the output of the algorithm) for the original instance (G, R). Regarding this solution, we can use the following two lemmas [15,32].

Lemma 10. *Let T^* be the final terminal-spanning tree constructed by the loss-contracting algorithm on an instance (G, R) and $loss^*$ the total cost of contracted edges. Then we have $c(T^*) \leq smt$ and $loss^* \leq loss \cdot \ln(\frac{mst-smt}{loss} + 1)$, where mst is the cost of an MST for R in the distance network of G, smt the cost of an SMT for (G, R), and $loss$ the sum of the losses of all FSTs in that SMT.*[2]

Lemma 11. *For quasi-bipartite graphs, $loss^* \leq 0.28 \cdot smt$ and the performance ratio of the loss-contracting algorithm is no worse than 1.28.*

Now we have the preliminaries to prove the following results.

Lemma 12. *Let T^* be the final terminal-spanning tree constructed by the loss-contracting algorithm and G^* the corresponding loss-contracted graph. Then, there is an (optimal) dual solution of DLP_{FSC} for G^* of value $c(T^*)$.*

Proof. The terminal-spanning tree T^* is an MST in the subgraph of G^* induced by the terminals. It is well known that for the MST problem (the special case of

[2] To keep the running time polynomial for general graphs, only FSTs with at most a constant number k of terminals are considered. In Lemma 10, smt is replaced by smt_k, the cost of an optimum Steiner tree under the restriction that all its full components contain at most k terminals. The performance ratios remain asymptotically the same, because it can be proved that $\frac{smt_k}{smt} \leq 1 + \frac{1}{1+\lfloor \log k \rfloor}$. See [32] for details.

the Steiner problem when all vertices are terminals), a pair of optimal primal and dual solutions $\hat{x}$ and $\hat{u}$ of the same value for LP_C resp. DLP_C can be constructed. This can be done by running a dual ascent algorithm and performing a reverse delete on the tight arcs, see [39,31] for details. In particular, the complementary slackness conditions can be satisfied, meaning that in the arc set of each cut $(\overline{W}, W)$ with $\hat{u}_W > 0$, there is exactly one arc $\boldsymbol{e}$ with $\hat{x}_{\boldsymbol{e}} = 1$ (these arcs represent a minimum spanning tree). Furthermore, it can be guaranteed that the sum of the cut variables corresponding to the edges on a path in the constructed MST (arcs $\boldsymbol{e}$ with $\hat{x}_{\boldsymbol{e}} = 1$) and separating the endpoints of the path is not larger than the cost of the longest edge on that path.
We claim that the dual solution $\hat{u}$ is also feasible for DLP_{FSC} w.r.t. G^*. Assume the contrary and let $\boldsymbol{T}$ (with root z_1 and leaves $z_2, \ldots, z_l$) be an FST with $\hat{U}_{\boldsymbol{T}} := \sum_{W,\ \boldsymbol{T} \in \Delta^-(W)} \hat{u}_W > c_{\boldsymbol{T}}$. Consider the (unique) path between z_1 and some z_i ($i \in \{2, \ldots, l\}$) in T^*. The sum of the relevant $\hat{u}$ values corresponding to the edges on this path is no greater than the cost of the longest edge on the path. Repeating this argument in proper order we establish that the sum of the $\hat{u}$ values for all considered (sub-)paths (which is an upper bound for $\hat{U}_{\boldsymbol{T}}$) is no larger than the cost of $l-1$ (different) edges on the corresponding paths. But then, introducing T and removing these edges would lead to a Steiner tree cheaper than T^*, a contradiction to the fact that $gain_{T^*}(T)$ is non-positive for any FST T. □

Lemma 13. *Let T^* be the final terminal-spanning tree constructed by the loss-contracting algorithm on an arbitrary graph G. Then, $v(LP_{FSC}) \geq c(T^*)$.*

Proof. By Lemma 12, there is a feasible solution $\hat{u}$ of value $c(T^*)$ for DLP_{FSC} w.r.t. G^*. Since G^* is obtained from G by setting some edge costs to zero, the corresponding FSTs in G have no smaller costs and $\hat{u}$ is also feasible w.r.t. G. □

Theorem 2. *For an instance (G, R) of the Steiner problem, let $loss^*$ be the total cost of the edges contracted by the loss-contracting algorithm and smt the cost of an optimal Steiner tree. Then, for the integrality gap of the relaxation LP_{FSC} we have: $\frac{v(LP_{FSC})}{v(P_{FSC})} \geq 1 - \frac{loss^*}{smt}$.*

Proof. From Lemma 13, we know that $v(LP_{FSC}) \geq c(T^*)$. On the other hand, T^* together with the contracted edges deliver a feasible solution for the original instance, so we have $c(T^*) + loss^* \geq smt = v(P_{FSC})$, and the claim follows. □

Lemma 14. *Let T^* be the final terminal-spanning tree constructed by the loss-contracting algorithm on a quasi-bipartite graph G. Then, $v(LP_C) \geq c(T^*)$.*

Proof. The claim follows immediately from Lemma 13 and Theorem 1. □

Theorem 3. *The integrality gap of the bidirected cut relaxation $(v(P_c)/v(LP_C))$ for a quasi-bipartite graph is smaller than* 1.39.

Proof. Using Lemma 14 and following the same argumentation as in the proof of Theorem 2, we get $\frac{v(LP_C)}{v(P_C)} \geq 1 - \frac{loss^*}{smt}$. From Lemma 11, we know that $\frac{loss^*}{smt} \leq 0.28$ for quasi-bipartite graphs, and the claim follows. □

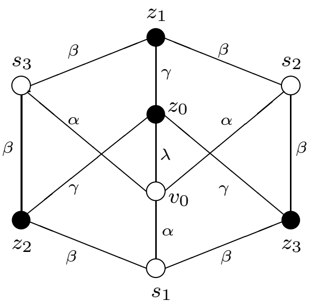


Fig. 4. Transfer from hypergraphs to graphs introduces an integrality gap

A natural question is whether a better bound (than the elementary bound 2) can be proved for this relaxation in the general case by the approach used here.

A fundamental difficulty is that Lemma 9 (and consequently Theorem 1) is not valid for general graphs (even after loss contraction), as Example 3 demonstrates. Therefore, a similar approach for general graphs should use a modified version of Lemma 9 that limits the gap introduced by the transfer from hypergraphs to graphs (at least for the worst case) properly.

Example 3. In the graph illustrated in Figure 4, let $\lambda = \alpha$, $\beta = 2\alpha$, and $\gamma = 3\alpha - \epsilon$ ($0 < \epsilon \ll \alpha$). For this graph, we have $v(LP_C) = 8\frac{1}{2}\alpha$ (setting x-variables to 1/2 for the arcs $[z_1, s_2]$,$[z_1, s_3]$,$[s_1, z_2]$,$[s_3, z_2]$,$[s_1, z_3]$,$[s_2, z_3]$, $[s_2, v_0]$,$[s_3, v_0]$,$[v_0, s_1]$ and to 1 for the arc $[v_0, z_0]$); while the optimal Steiner tree, which is in this case identical to the minimum terminal-spanning tree, consists of the edges $(z_0, z_1), (z_0, z_2), (z_0, z_3)$ and has the cost $3\gamma = 9\alpha - 3\epsilon$. The loss-contracting algorithm starts and ends with this terminal-spanning tree, without contracting any edges. On the other hand, it can be observed that for any choice of FSTs F such that $G = G_F$, the relaxation LP_{FSC} yields the value $3\gamma = 9\alpha - 3\epsilon$ for the hypergraph (R, F). Intuitively, this is because that the re-usage of the arc $[v_0, s_1]$ must be "paid for" in the case of LP_{FSC}, in contrast to the case of LP_C.

5 Concluding Remarks

Despite the achievements described in this article, of course many challenges are open. Here, we mention just a few which we consider as specially important.

- Further study of the structural properties of relaxations can lead both to new theoretical insights and algorithmic developments. In particular, the question of the integrality gap of the cut relaxation is quite intriguing.
- The algorithmic potential of full Steiner trees is presumably not exhausted. Natural candidates for further exploitation are special metrics (not only geometric ones) as they arise in certain applications (like the computation of phylogenetic trees).
- Finally, the transfer of methods that have proved to be useful in this context to similar problems is a promising (but in some cases non-trivial) task.

References

1. Althaus, E., Polzin, T., Vahdati Daneshmand, S.: Improving linear programming approaches for the Steiner tree problem. In: Jansen, K., Margraf, M., Mastrolli, M., Rolim, J.D.P. (eds.) WEA 2003. LNCS, vol. 2647, pp. 1–14. Springer, Heidelberg (2003)
2. Aneja, Y.P.: An integer linear programming approach to the Steiner problem in graphs. Networks 10, 167–178 (1980)
3. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. Mathematical Programming 97, 91–153 (2003)
4. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: fast subset convolution. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, pp. 67–74 (2007)
5. Chakrabarty, D., Devanur, N.R., Vazirani, V.V.: New geometry-inspired relaxations and algorithms for the metric Steiner tree problem. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) IPCO 2008. LNCS, vol. 5035, pp. 344–358. Springer, Heidelberg (2008)
6. Cheng, X., Du, D.-Z. (eds.): Steiner Trees in Industry. Combinatorial Optimization, vol. 11. Kluwer, Dordrecht (2001)
7. Chopra, S., Rao, M.R.: The Steiner tree problem I: Formulations, compositions and extension of facets. Mathematical Programming 64, 209–229 (1994)
8. Chopra, S., Tsai, C.-Y.: Polyhedral approaches for the Steiner tree problem on graphs. In: Cheng, X., Du, D.-Z. (eds.) Steiner Trees in Industry. Combinatorial Optimization, vol. 11, pp. 175–202. Kluwer, Dordrecht (2001)
9. Cieslik, D.: Steiner minimal tress. Kluwer, Dordrecht (1998)
10. Du, D.-Z., Smith, J.M., Rubinstein, J.H. (eds.): Advances in Steiner Trees. Kluwer, Dordrecht (2000)
11. Duin, C.W.: Steiner's Problem in Graphs. Ph.D thesis, Amsterdam University (1993)
12. Fuchs, B., Kern, W., Mölle, D., Richter, S., Rossmanith, P., Wang, X.: Dynamic programming for minimum Steiner trees. Theory of Computing Systems 41(3), 493–500 (2007)
13. Goemans, M.X.: Personal communication (1998)
14. Goemans, M.X., Bertsimas, D.J.: Survivable networks, linear programming relaxations and the parsimonious property. Mathematical Programming 60, 145–166 (1993)
15. Gröpl, C., Hougardy, S., Nierhoff, T., Prömel, H.J.: Approximation algorithms for the Steiner tree problem in graphs. In: Cheng, X., Du, D.-Z. (eds.) Steiner Trees in Industry. Combinatorial Optimization, vol. 11, pp. 235–280. Kluwer, Dordrecht (2001)
16. Hwang, F.K., Richards, D.S., Winter, P.: The Steiner Tree Problem. Annals of Discrete Mathematics, vol. 53. North-Holland, Amsterdam (1992)
17. Jain, K., Mahdian, M., Salavatipour, M.R.: Packing Steiner trees. In: Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms, pp. 266–274 (2003)
18. Koch, T., Martin, A.: Solving Steiner tree problems in graphs to optimality. Networks 32, 207–232 (1998)
19. Koch, T., Martin, A., Voß, S.: Steinlib: An updated library on Steiner tree problems in graphs. In: Cheng, X., Du, D.-Z. (eds.) Steiner Trees in Industry. Combinatorial Optimization, vol. 11, pp. 285–326. Kluwer, Dordrecht (2001)

20. Könemann, J., Konjevod, G., Parekh, O., Sinha, A.: Improved approximations for tour and tree covers. In: Jansen, K., Khuller, S. (eds.) APPROX 2000. LNCS, vol. 1913, pp. 184–193. Springer, Heidelberg (2000)
21. Magnanti, T.L., Wolsey, L.A.: Optimal Trees. In: Ball, M.O., et al. (eds.) Handbooks in Operations Research and Management Science, ch. 9, vol. 7. Elsevier, Amsterdam (1995)
22. Poggi de Aragão, M., Werneck, R.F.: On the implementation of MST-based heuristics for the Steiner problem in graphs. In: Mount, D.M., Stein, C. (eds.) ALENEX 2002. LNCS, vol. 2409, pp. 1–15. Springer, Heidelberg (2002)
23. Polzin, T., Vahdati Daneshmand, S.: Algorithmen für das Steiner-Problem. Master's thesis, Universität Dortmund (1997)
24. Polzin, T., Vahdati Daneshmand, S.: Primal-Dual Approaches to the Steiner Problem. In: Jansen, K., Khuller, S. (eds.) APPROX 2000. LNCS, vol. 1913, pp. 214–225. Springer, Heidelberg (2000)
25. Polzin, T., Vahdati Daneshmand, S.: A comparison of Steiner tree relaxations. Discrete Applied Mathematics 112, 241–261 (2001)
26. Polzin, T., Vahdati Daneshmand, S.: Improved algorithms for the Steiner problem in networks. Discrete Applied Mathematics 112, 263–300 (2001)
27. Polzin, T., Vahdati Daneshmand, S.: Extending reduction techniques for the Steiner tree problem. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 795–807. Springer, Heidelberg (2002)
28. Polzin, T., Vahdati Daneshmand, S.: On Steiner trees and minimum spanning trees in hypergraphs. Operations Research Letters 31(1), 12–20 (2003)
29. Polzin, T., Vahdati Daneshmand, S.: Practical partitioning-based methods for the Steiner tree problem. In: Àlvarez, C., Serna, M. (eds.) WEA 2006. LNCS, vol. 4007, pp. 241–252. Springer, Heidelberg (2006)
30. Prömel, H.-J., Steger, A.: The Steiner Tree Problem: A Tour through Graphs, Algorithms and Complexity. Vieweg (2002)
31. Rajagopalan, S., Vazirani, V.V.: On the bidirected cut relaxation for the metric Steiner tree problem. In: Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms, pp. 742–751 (1999)
32. Robins, G., Zelikovsky, A.: Improved Steiner tree approximation in graphs. In: Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms, pp. 770–779 (2000)
33. SteinLib, `http://elib.zib.de/steinlib/steinlib.php`
34. Uchoa, E., de Aragão, M.P., Ribeiro, C.C.: Preprocessing Steiner problems from VLSI layout. Networks 40, 38–50 (2002)
35. Vahdati Daneshmand, S.: Algorithmic Approaches to the Steiner Problem in Networks. Ph.D thesis, University of Mannheim (2004), `http://madoc.bib.uni-mannheim.de/madoc/volltexte/2004/176`
36. Voß, S.: Steiner-Probleme in Graphen. Hain-Verlag, Frankfurt/M. (1990)
37. Warme, D.M.: Spanning Trees in Hypergraphs with Applications to Steiner Trees. Ph.D thesis, University of Virginia (1998)
38. Warme, D.M., Winter, P., Zachariasen, M.: Exact algorithms for plane Steiner tree problems: A computational study. In: Du, D.-Z., Smith, J.M., Rubinstein, J.H. (eds.) Advances in Steiner Trees, pp. 81–116. Kluwer, Dordrecht (2000)
39. Wong, R.T.: A dual ascent approach for Steiner tree problems on a directed graph. Mathematical Programming 28, 271–287 (1984)

A Survey on Multiple Objective Minimum Spanning Tree Problems

Stefan Ruzika and Horst W. Hamacher

Department of Mathematics, University of Kaiserslautern, P.O. Box 3049, D-67653
Kaiserslautern, Germany
{ruzika,hamacher}@mathematik.uni-kl.de

Abstract. We review the literature on minimum spanning tree problems with two or more objective functions (MOST) each of which is of the sum or bottleneck type. Theoretical aspects of different types of this problem are summarized and available algorithms are categorized and explained. The paper includes a concise tabular presentation of all the reviewed papers.

1 Introduction

Let $G = (V, E)$ be a (undirected, simple) graph with vertex set $V = \{v_1, \dots v_n\}$ and edge set $E = \{e_1, \dots, e_m\}$ where $e_j = (v_{i_1}, v_{i_2})$ is an (unordered) pair of vertices. Let $c\colon E \to \mathbb{R}^p$ be a vector-valued cost function on the edge set. A *spanning tree of* G is a connected, acyclic subgraph $T \sqsubseteq G$, $T = (V(T), E(T))$, with vertex set $V(T) = V$. Let $\mathcal{T}$ denote the *set of spanning trees* of a given graph G. The *multiple objective minimum spanning tree problem* is defined as

$$\text{(MOST)} \qquad \begin{aligned} &\min\ f(T) = (f_1(T), \dots, f_p(T)) \\ &\text{s.t.}\ \ T \in \mathcal{T}. \end{aligned}$$

In contrast to the (single objective) spanning tree problem, the objective function $f\colon \mathcal{T} \to \mathbb{R}^p$ is vector-valued, i.e., composed of component functions $f_k\colon \mathcal{T} \to \mathbb{R}$. The components f_k of f are *sum objectives*, i.e.

$$f_k = \sum_{e \in E(T)} c^k(e),$$

or *bottleneck objectives*, i.e.

$$f_k = \max_{e \in E(T)} c^k(e).$$

In the following, we assume sum objectives unless stated otherwise. In MOST, the task is to find optimizers which simultaneously minimize the k component functions. Since the notion of optimality is not canonical in $\mathbb{R}^p$ (note that there is no canonical order in $\mathbb{R}^p$), an optimality concept has to be defined. In this paper, we use the Edgeworth-Pareto optimality definition. We define the *set of efficient spanning trees*

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 104–116, 2009.

$$\begin{aligned} X_E := \{T \in \mathcal{T} : \nexists\, \bar{T} \in \mathcal{T} &\text{ with } f_k(\bar{T}) \leq f_k(T) \text{ for all } k \leq p \\ &\text{and } f_j(\bar{T}) < f_j(T) \text{ for at least one } j \leq p\} \end{aligned}$$

and the *set of weakly efficient spanning trees*

$$X_{wE} := \{T \in \mathcal{T} : \nexists\, \bar{T} \in \mathcal{T} \text{ with } f_k(\bar{T}) < f_k(T) \text{ for all } k \leq p\}$$

The images of these sets under the objective function f are called the *set of nondominated points* $Y_N := f(X_E)$ and the *set of weakly nondominated points* $Y_{wN} := f(X_{wE})$. Note that in single objective optimization, the task is to find a single minimizer of the scalar-valued objective function. In contrast, minimizing the vector-valued objective function f is understood as finding (a subset of) the set Y_N and for each of these nondominated points at least one efficient spanning tree. For an introduction to single objective spanning tree problems and general multiple objective optimization, we refer to the books of Ahuja, Magnanti, and Orlin [2] and Ehrgott [10].

Efficient solutions and nondominated points are often computed by *scalarization methods* which transfer the multiple objective programming problem into a single criterion, i.e. scalar-valued optimization problem. In the following we state some of these scalarization methods in the context of MOST.

The *weighted-sum problem* can be stated as

$$\begin{array}{lll} \text{WSP}(\lambda) & \min & \displaystyle\sum_{i=1}^{p} \lambda_i f_i(T) \\ & \text{s.t.} & T \in \mathcal{T}. \end{array}$$

where $\lambda_i > 0$, $i = 1 \ldots p$, is a weight associated with objective i. It is easy to see that optimal spanning trees of WSP(λ) are efficient. Efficient spanning trees that can be obtained by solving WSP(λ) for any choice of $\lambda \in \mathbb{R}^p$ are called *supported* while all remaining efficient trees are called *unsupported.* The corresponding vectors in the objective space are analogously called *(un)supported nondominated points.* Supported efficient spanning trees are located on the boundary of the convex hull of Y_N. For sum objectives, WSP(λ) is a (single objective) spanning tree problem and can be solved by the well-known standard techniques.

Another scalarization is the *ε-constraint method* which is defined by a parameter $\varepsilon \in \mathbb{R}^{p-1}$ restricting the feasible set of the single objective minimum spanning tree problem.

$$\begin{array}{lll} \text{EPS}(\varepsilon) & \min & f_i(T) \\ & \text{s.t.} & T \in \mathcal{T} \\ & & f_j(T) \leq \varepsilon_j, \quad j \neq i. \end{array}$$

Optimal spanning trees for EPS(ε) are weakly efficient and any efficient tree can be computed by solving EPS(ε) for a certain choice of ε. For biobjective sum problems, EPS(ε) is also known as the *weight-constraint spanning tree problem* which is treated in [1], [14], [18], [20], and [36],. Since the latter problem is known to be NP-hard, the ε-constraint minimum spanning tree problem is NP-hard as well.

In the next sections, we review the current literature and present some new results classified according to the number of sum objectives in MOST. While Sections 3 and 4 deal with at least 2 sum objectives with regard to theory and algorithms, respectively, Section 2 considers the special case of at most one sum objective. We conclude the paper by a tabular respresentation of the reviewed papers in Section 6.

2 MOST with at Most One Sum Objective

In this section we consider 1 sum objective function and thus $p-1$ bottleneck objective functions. Since there are m edges, each bottleneck objective can take on at most m different values. Consequently, the number of nondominated points is bounded by $\mathcal{O}(m^{p-1})$.

Theorem 1. *If the objective function of MOST consists of* $p-1$ *bottleneck and* 1 *sum objective function, then* $|Y_N| \in \mathcal{O}(m^p)$.

Sergienko and Perepelitsa [35] consider minimum spanning tree problems with two parameters on each edge, one bottleneck and one sum objective function. They propose a polynomial time algorithm for finding the nondominated set. Their algorithm temporarily disregards the bottleneck objective and solves a single objective minimum spanning tree problem. In the corresponding solution the maximum bottleneck parameter is identified and all edges in the graph having maximum bottleneck parameter are deleted. The reduced graph is processed likewise until a feasible solution can no longer be found.

Melamed and Sigal [23,24,25,26] deal with spanning tree problems having one sum objective and (at least) one bottleneck objective function. All papers numerically investigate the fraction of supported spanning trees among the set of efficient spanning trees. For this purpose all nondominated points are identified using some enumeration which explains the relatively small test instances. In the four experimental studies, random parameters are always chosen independently and uniformly distributed in the intervals $[0, 20], [0, 100]$ or $[0, 500]$. The sizes of the graphs vary between $n = 10$ and $n = 150$. Depending on the specific setup, Melamed and Sigal report values averaged over 20 to 50 instances.

In [23] two biobjective problems are addressed: The first having one bottleneck and one sum objective function, the second having two bottleneck objective functions. The algorithm used to find the nondominated set is similar to the algorithm of Sergienko and Perepelitsa [35]. Melamed and Sigal conclude that the fraction of supported spanning trees decreases with increasing

- size of the cost interval [23],
- number of vertices in the graph [23], and
- number of efficient spanning trees [24].

Melamed and Sigal extend their work to the tri-objective case of either having two bottleneck and one sum objective function or having three bottleneck objective functions (cf. [25,26]). The idea of the algorithm in [35] is adapted to having at least two bottleneck objective functions: Maxima found in previous iterations are excluded consecutively by changing the costs of the edges thus yielding new candidate points. If no feasible solution can be found, the set of candidates is filtered for dominated solutions. Interestingly, Melamed and Sigal report that the fraction of supported efficient spanning trees is much greater for these problems than for the corresponding biobjective problems. Second, the cardinality of the efficient set is increasing rapidly (the precise factors depend on the specific problem setup) when going from two to three objectives.

3 MOST with at Least Two Sum Objectives - Theory

In this section, we consider MOST with sum objectives only. A multiple objective combinatorial optimization problem is called *intractable* if the cardinality of the set of optimal values, i.e., $|Y_N|$, can be exponential in the size of the instance (see [10]). MOST is intractable and NP-hard, even for $p = 2$ objective functions.

Theorem 2 (Camerini, Galbiati, and Maffioli [4]). *MOST is NP-hard.*

Theorem 3 (Hamacher and Ruhe [17]). *MOST is intractable.*

Proof. Consider the complete graph $G = K_n$ of n vertices and $m = \frac{n(n-1)}{2}$ edges. G contains n^{n-2} spanning trees. For each edge e_i we define its costs as

$$\begin{aligned} c^1(e_i) &:= 2^{i-1} \\ c^2(e_i) &:= 2^m - 2^{i-1} = 2^m - c^1(e_i). \end{aligned}$$

It is $c^1(e_i) + c^2(e_i) = 2^m$ for all $i \in \{1, \dots, m\}$ and $c^1(T) + c^2(T) = (n-1)2^m$ for all $T \in \mathcal{T}$. All pairs of spanning trees $T_1, T_2 \in \mathcal{T}$, $T_1 \neq T_2$, are incomparable by the uniqueness of the number representation of the binary system. Thus, $|Y_N| = n^{n-2}$. □

There is a one-to-one correspondence between efficient spanning trees and nondominated solutions in the example of Hamacher and Ruhe [17]. Since there are n^{n-2} many efficient spanning trees, the following corollary can be immediately derived from Theorem 3.

Corollary 1. *The set of supported efficient spanning trees and the set of nondominated points is, in general, exponentially large in the number of vertices of G.*

In the example used in Theorem 3 and Corollary 1 all efficient spanning trees are optimal for a single weighted sum problem since $c^1(T)+c^2(T) = (n-1)2^m$ for all $T \in \mathcal{T}$. The corresponding nondominated points are located on a straight line in Y. They are all supported. Consequently, the nondominated frontier has no (intermediate) breakpoints for this example. In general, this number of breakpoints is polynomially bounded in the number of vertices.

Theorem 4. *The number of breakpoints of the nondominated frontier is polynomially bounded for $p = 2$.*

Proof. Each breakpoint can be computed with the weighted sum method for some value of $\lambda \in (0, 1)$. The weighted sum method can be realized with a greedy algorithm for single objective spanning tree problems. For each edge $j \in E$ and $\lambda \in (0, 1)$, the costs of this single objective problem are given as

$$\lambda c_j^1 + (1 - \lambda)c_j^2 = c_j^2 + (c_j^1 - c_j^2)\lambda.$$

For each $j \in E$, we may think of $c_j^2 - (c_j^1 - c_j^2)\lambda$ as a function of λ on the interval $(0, 1)$. These $|E| = m$ many linear functions have $k \leq m^2$ points of intersection at the λ-values $0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_k \leq 1$. For $\lambda \in [\lambda_i, \lambda_{i+1}]$, $i \in \{1, \ldots, k-1\}$, the sorting of the linearly combined costs remains the same. Consequently the greedy algorithm returns for any such λ the same solution. Since the sorting of the edges changes at most m^2-times, the greedy algorithm returns at most m^2 different solutions when λ varies over $(0, 1)$. Thus, there are at most m^2-many breakpoints. □

In order to compute all nondominated solutions with a 2-phase algorithm, Theorem 4 implies that the first phase of computing $(\text{conv}(Y))_N$ can be accomplished in a polynomial number of applications of the weighted sum method.

The question about the cardinality of the set of unsupported nondominated points arises directly in the context of the preceding intractability results.

Theorem 5. *The set of unsupported efficient spanning trees might be exponentially large in the number of vertices of G.*

Proof. Consider the complete graph $G = K_n$. For each edge e_i we define its costs as

$$\begin{aligned} c^1(e_i) &:= 2^{i-1} \\ c^2(e_i) &:= 2^{m+2} - 2^{i-1} = 2^{m+2} - c^1(e_i) \end{aligned}$$

The same reasoning as in the proof of Theorem 3 holds true here as well. Note that $|c^i(T_1) - c^i(T_2)| > 4$, $i = 1, 2$, for every pair $T_1, T_2 \in \mathcal{T}$, $T_1 \neq T_2$. We slightly modify the graph now by introducing two new vertices v_{n+1} and v_{n+2} and three new edges with the following costs:

$$\begin{aligned} e_{m+1} &:= (v_{n+1}, v_{n+2}) \text{ with } c(e_{m+1}) = (0, 0) \\ e_{m+2} &:= (v_1, v_{n+1}) \text{ with } c(e_{m+2}) = (1, 0) \\ e_{m+3} &:= (v_1, v_{n+2}) \text{ with } c(e_{m+3}) = (0, 2) \end{aligned}$$

This new graph has $3n^{n-2}$ spanning trees and $2n^{n-2}$ efficient spanning trees. Among those, $n^{n-2} + 1$ are supported, while $n^{n-2} - 1$ are nonsupported. □

An interesting question relates to results about adjacency of efficient spanning trees. In the remainder of this section we elaborate on this issue for problems with $p = 2$.

Definition 1. *Two trees are called* adjacent *if they have $n-2$ arcs in common.*

Alternatively, two trees T_1 and T_2 are adjacent if there is some edge $e \in E(T_1)$ and some edge $f \notin E(T_1)$ such that $E(T_2) = E(T_1) \setminus \{e\} \cup \{f\}$. This edge exchange is called *edge swap* or simply *swap*. Research on adjacency of efficient solutions is motivated by the fact that the set of optimal solutions of the minimal spanning tree problem is connected:

Theorem 6. *Let T_1 and T_2 denote two minimal spanning trees. Then there is a sequence of edge swaps yielding T_2 from T_1. Moreover, a sequence can be found such that all spanning trees constructed in this sequence are minimal.*

This result allows an efficient enumerative search to obtain all optimal solutions to the single objective spanning tree problem. Surprisingly, for MOST, the set of efficient spanning trees is non-connected.

Definition 2. *The adjacency graph of (weakly) efficient spanning trees $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by the vertex set $\mathcal{V} = X_E$. Two vertices v_i and v_j are connected by an (undirected) edge (v_i, v_j) if their corresponding trees are adjacent.*

Ehrgott and Klamroth [12] provide an instance of the multiple objective minimum spanning tree problem having a non-connected adjacency graph of efficient spanning trees. This implies that it is not possible to swap from one efficient tree to another while visiting efficient trees only. Gorski, Klamroth and Ruzika [15] extend these results.

Theorem 7. *The adjacency graph of (weakly) efficient spanning trees is non-connected in general. The number of connected components of $\mathcal{G}$ and the cardinality of the components of $\mathcal{G}$ may grow exponentially in the size of the input data.*

These results imply that it is, in general, not enough to swap to get all efficient spanning trees. Ruzika [32] proved some structural results about the set of supported efficient spanning trees and suggested their usage in terms of a branch and bound algorithm for the constrained spanning tree problem. Henn [19] showed the efficiency of this algorithm.

4 MOST with at Least Two Sum Objectives - Algorithms

Corley [6] intended to compute X_E for MOST, however, Hamacher and Ruhe [17] demonstrated that Corley's algorithm is only capable of computing a subset of X_E. Corley suggests to iteratively compose spanning trees similar to Prim's idea. In each iteration the set of vertices already contained in the subtree defines a cut in the graph. The subtree is expanded by all nondominated edges along this cut. In contrast to the single objective case a set of efficient extensions has to be considered and, consequently, a set of subtrees has to be processed.

Hamacher and Ruhe [17] address the theory of multiple objective minimum spanning tree problems (see Theorem 3) and provide an algorithm for the biobjective case. Their own algorithm is capable of computing the set of supported efficient spanning trees exactly. Alternatively—and this is their main focus—their two-phase method can compute a well-dispersed set of feasible spanning trees serving as a representation of X_E. The first phase is realized by a dichotomous search scheme. First, the lexicographical optimal spanning trees are computed. Given two extremal spanning trees, a weighted sum problem is solved to determine either a new extremal efficient spanning tree in between the two given trees or to decide that there is no such solution. After the first phase is finished a neighborhood search is employed in the second phase to find unsupported solutions closing this gap. These solutions are guaranteed to be locally efficient.

Andersen, Jörnsten, and Lind [3] numerically compare two heuristics for the biobjective MOST. In both heuristics the set of supported nondominated solutions is found in the first phase by some algorithm, e.g., the algorithm in [17]. Although being aware of the fact that the set of efficient solutions is not connected, the authors suggest to exploit adjacency to construct an approximation of the nondominated set. Two different methods are proposed to generate unsupported trees: A neighborhood search and the "adjacent search". In the neighborhood search spanning trees being adjacent (in the sense of [12]) to at least one previously found (locally) efficient spanning tree are investigated. In the adjacent search, a new heuristic, only those spanning trees are constructed which are adjacent to at least two (locally efficient) spanning trees. Two methods are proposed for generating spanning trees which are adjacent to at least two given spanning trees. The two proposed heuristics are tested on various instances. On average, adjacent search investigates less candidate points than neighborhood search. The computational effort in constructing new candidate solutions is, however, larger for the adjacent search.

A two-phase procedure for the biobjective minimum spanning tree problem is proposed by **Ramos, Alonso, Sicilia, and González** [31]. Their procedure does not only find one efficient solution for each nondominated points, but all efficient solutions. For this purpose an enumerative procedure for finding all minimum spanning trees of a single objective minimum spanning tree problem is developed. The first phase algorithm for the biobjective problem is similar to the algorithm of [17]. However, given two consecutive extreme points, all efficient solutions yielding the same nondominated point are computed by the single objective enumeration algorithm. Unsupported efficient solutions are found by a branch and bound method. A search tree having n levels, from the root 0 to the lowest level $n-1$, is constructed and passed in depth first order. Each vertex x of this search tree corresponds to an edge of the adjacency list of vertex $i \in V$. Furthermore, with each node x a partial solution (a forest) is associated. This solution corresponds to all edges on the path from the root of the search node to node x. Branching is performed for each edge in the adjacency list of vertex $i+1$. Those edges which are already part of the partial solution or those which lead to a cycle in the partial solution are neglected. For each partial solution a lower bound on the cost vector is computed. If this lower bound is dominated by some previously found solution, the branch corresponding to this solution is fathomed. Solutions found by this branch and bound method do not have to be efficient. Thus, an additional filtering procedure has to be applied excluding the dominated solutions. The performance of this two-phase algorithm is numerically investigated. Interestingly, the running time decreases with an increasing number of vertices. Ramos, Alonso, Sicilia, and González note that the number of unsupported efficient solutions decreases for their test set-up with an increasing number of vertices.

The numerical study of **Steiner and Radzik** [37] compares two different two-phase methods for the biobjective integer minimum spanning tree problem. For the first phase the algorithm of Hamacher and Ruhe [17] is utilized to find the set of supported efficient solutions for both approaches. In the second phase a k-best minimum spanning tree algorithm is compared to a branch and bound algorithm. In the former variant the triangles between all pairs of consecutive nondominated extreme points are searched for unsupported efficient trees by the (single objective) k-best minimum spanning tree algorithm of Gabow [13]. In order to obtain a single cost objective, the two cost components are aggregated by a weighted sum. The algorithm of Gabow is then applied with this cost function. Each solution is used to reduce the search space by splitting the original rectangle into two smaller ones. This k-best approach is compared with the branch and bound procedure of Ramos, Alonso, Sicilia, and González [31] in numerical tests. Steiner and Radzik conclude that the k-best algorithm substantially outperforms the branch and bound method due to the huge search tree and rather ineffective bounding procedure of the second approach.

5 Related Topics

There is some literature adressing topics which are closely related to MOST.

Hutson and ReVelle [21,22] assume a tree graph and two arc parameters - referred to as costs and coverage - to be given. They introduce the covering problem on tree graphs as the problem of finding a connected subset of arcs minimizing costs and maximizing coverage. An integer programming formulation of this biobjective spanning tree problem is introduced and a few example problems are solved using an all-purpose solver. The articles do not contain an algorithm nor do they discuss theoretical properties of the biobjective spanning tree problem. The main goal is the transfer of coverage concepts mainly used in facility location to network analysis.

Schweigert [33,34] recognizes that the weighted sum method delivers efficient spanning trees and formulates an algorithm for finding all supported efficient spanning trees similar to the method of [17]. It is also shown that for any unsupported efficient spanning tree there exists a total ordering of the set of arcs such that the tree is optimal for Kruskal's algorithm. Unfortunately, the proof of this theorem assumes the unsupported tree to be given and then constructs the total ordering. Consequently, a practical algorithm for finding unsupported spanning trees cannot be derived from this observation.

Perny and Spanjaard [28] consider extensions of combinatorial optimization problems: Preferences are defined by quasi-transitive relations on the set of feasible solutions. They formally introduce preference-based combinatorial optimization problems and general algorithms. These algorithms find the set of preferred solutions only if the preference relation satisfies some independence axiom. The idea of preference relations is specialized to the biobjective minimum spanning tree problem as follows: A tree T_1 is preferred to T_2 if and only if the weighted augmented Tchebycheff metric of T_1 is less than that of T_2. Unfortunately, this relation does not satisfy the independence axiom. Hence, the proposed algorithms do not find the set of preferred solutions exactly, but an approximation.

Minoux [27] considers general problems over a finite ground set E. Two cost values are associated with each element in E. An algebraic sum of bottleneck and linear cost function constitutes the single objective function. It is assumed that there exists a polynomial-time solution algorithm for the same problem with linear cost objective function only. It is observed that the max-term can take on at most $|E|$ values. If the optimal solution value v for the bottleneck objective function was given prior to the optimization by some oracle, then the sum objective function would only have to be optimized over all elements of the ground set not exceeding v in the bottleneck objective (it should be remarked that this idea is also used in [23,24,25,26] and [35]). Based on this observation a solution algorithm is proposed. In the k-th iteration of the algorithm the sum-problem is solved over a modified ground set E^k. In E^k only the k smallest elements with respect to the bottleneck objective are included. A solution to this modified problem is also a solution to the problem with combined objective function. If no solution is found, k is increased. After at most $|E|$ solutions of sum-problems the optimal solution to the combined problem is found. Several improvements of this algorithm are discussed in the paper as well as the application to a variety of problems (also the spanning tree problem). There are several follow-ups (e.g. [9], [16], [29], [30]) for the same problem presenting new algorithms or exploiting more efficient data structures for similar algorithmic ideas. **Dell'Amico and Maffioli** [8] study the complexity of 56 problems having an algebraic sum of a linear and a non-linear function as objective function.

Dell'Amico and Maffioli [7] consider arborescence problems with two cost parameters on each arc. Six different elementary objective functions (e.g., bottleneck, minimum sum, or sum of cost of the longest path objective function) are proposed and always two of them are linearly combined, thus, yielding a scalar valued objective function. The resulting problems are classified as NP-hard or polynomial time algorithms solving the problem exactly are given.

6 Conclusion

We have reviewed known results and algorithms for the multiple objective minimum spanning tree problem and related topics. All papers included in our review are concisely listed in Table 3 using the classification scheme for multiple objective combinatorial optimization problems of Ehrgott and Gandibleux [11].

Table 1. Entries for *Problem Type*

Entry	Explanation
E	Finding the efficient set
e	Finding a subset of the efficient set
SE	Finding the supported efficient solutions
MO	Solving the max ordering problem
C	Finding a compromise solution

Table 2. Entries for *Solution Method*

Entry	Explanation
SP	Exact algorithm specifically designed for the problem
BB	Algorithm based on branch and bound
conv	Method for computing nondominated frontier based on iterative weighted sum problems
NS	Neighborhood search
Prim	Prim-type algorithm
rank	Ranking method
k-best	K-best spanning tree procedure

Table 3 has six columns. In the first column, we refer to the paper under consideration. The number of objective functions in this paper is listed in the second column. Here, we distinguish between approaches designed for biobjective (denoted by 2) and general multiple objective problems (denoted by p). Note that all papers consider sum objectives. The third and fourth column provide information about the type of the problem and the applied solution methods, respectively. Tables 1 and 2 explain the abbreviations used in these columns. In the last two columns, a "+" indicates that examples are included in the paper and that the algorithm has been implemented by the authors, respectively. The abbreviations "sum" and "max" indicate in Table 3 the type of objective function. It should be noted that the articles of Chen, Guo, Tu, and Chen [5], Dell'Amico and Maffioli [7,8], Perny and Spanjaard [28], and Schweigert [33,34] are not listed since they are not discussed in the survey or do not fit this algorithm-oriented scheme.

Almost all algorithms reviewed in the previous sections address problems with two objectives. Some algorithmic ideas are prevalent in the surveyed literature. They seem to be widely accepted and they are frequently used like the computation of the nondominated frontier (cf. the approaches using the algorithm of [17]) or the treatment of bottleneck objective functions (cf. [35] and [23,24,25,26]). The methods for identifying unsupported efficient spanning trees vary and include neighborhood searches [17], branch and bound procedures [31], and k-best spanning tree algorithms [37]. The question of adjacency of efficient spanning trees is not well understood. All known neighborhood structures for the multiple objective minimum spanning tree problem lead to a non-connected adjacency graph.

Table 3. Classification for the reviewed papers

Paper	Number and Type of Objectives	Problem Type	Solution Method	Example	Implemented
[3]	2 sum	SE, e	conv, NS	+	+
[6]	p sum	e	Prim	-	-
[17]	2 sum	SE, e	conv, NS	+	-
[17]	p sum	MO	rank	+	-
[21,22]	2 sum	-	-	+	-
[23,24]	1 sum, 1 max	E	SP	+	+
[25,26]	1 sum, 2 max	E	SP	-	+
[31]	2 sum	E	conv, BB	+	+
[35]	1 sum, 1 max	E	SP	-	-
[37]	2 sum	E	conv, k-best / BB	+	+
[9,16]	sum+max	C	SP	-	-
[27,29,30]	sum+max	C	SP	-	-

Acknowledgement. The authors thankfully acknowledge partial support from the Deutsche Forschungsgemeinschaft (DFG) grant HA 1737/7 as part of the Schwerpunktprogramm "Algorithmik großer und komplexer Netzwerke".

References

1. Aggarwal, V., Aneja, Y.P., Nair, K.P.K.: Minimal spanning tree subject to a side constraint. Computers and Operations Research 9(4), 287–296 (1982)
2. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Inc., Englewood Cliffs (1993)
3. Andersen, K.A., Joernsten, K., Lind, M.: On bicriterion minimal spanning trees: an approximation. Computers and Operations Research 23, 1171–1182 (1996)
4. Camerini, P.M., Galbiati, G., Maffioli, F.: The complexity of multi-constrained spanning tree problems. In: Lovasz, L. (ed.) Theory of Algorithms, pp. 53–101. North-Holland, Amsterdam (1984)
5. Chen, G., Guo, W., Tu, X., Chen, H.: An improved algorithm to solve the multicriteria minimum spanning tree problem. J. Softw. 17(3), 364–370 (2006)
6. Corley, H.W.: Efficient spanning trees. Journal of Optimization Theory and Applications 45(3), 481–485 (1985)
7. Dell'Amico, M., Maffioli, F.: On some multicriteria arborescence problems: Complexity and algorithms. Discrete Applied Mathematics 65, 191–206 (1996)
8. Dell'Amico, M., Maffioli, F.: Combining linear and non-linear objectives in spanning tree problems. Journal of Combinatorial Optimiation 4(2), 253–269 (2000)
9. Duin, C.W., Volgenant, A.: Minimum deviation and balanced optimization: A unified approach. Operations Research Letters 10, 43–48 (1991)
10. Ehrgott, M.: Multicriteria Optimization, 2nd edn. Springer, Berlin (2005)
11. Ehrgott, M., Gandibleux, X.: Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys. Kluwer Academic Publishers, Boston (2002)
12. Ehrgott, M., Klamroth, K.: Connectedness of efficient solutions in multiple criteria combinatorial optimization. European Journal of Operational Research 97, 159–166 (1997)

13. Gabow, H.N.: Two algorithms for generating weighted spanning trees in order. SIAM Journal on Computing 6, 139–150 (1977)
14. Goemans, M.X., Ravi, R.: A constrained minimum spanning tree problem. In: Proccedings of the Scandinavian Workshop on Algorithmic Theory (SWAT), pp. 66–75. Springer, Heidelberg (1996)
15. Gorski, J., Klamroth, K., Ruzika, S.: Connectedness of efficient solutions in multiple objective combinatorial optimization. Report in Wirtschaftsmathematik, Nr. 102/2006, University of Kaiserslautern, Department of Mathematics (2006) (submitted for publication)
16. Gupta, S.K., Punnen, A.P.: Minimum deviation problems. Operations Research Letters 7, 201–204 (1988)
17. Hamacher, H.W., Ruhe, G.: On spanning tree problems with multiple objectives. Annals of Operations Research 52, 209–230 (1994)
18. Hassin, R., Levin, A.: An efficient polynomial time approximation scheme for the constrained minimum spanning tree problem using matroid intersection. SIAM Journal on Computing 33(2), 261–268 (2004)
19. Henn, S.: The weight-constrained minimum spanning tree problem. Master's thesis, Technische Universität Kaiserslautern (2007)
20. Hong, S.P., Chung, S.J., Park, B.H.: A fully polynomial bicriteria approximation scheme for the constrained spanning tree problem. Operations Research Letters 32, 233–239 (2004)
21. Hutson, V.A., ReVelle, C.S.: Maximal direct covering tree problem. Transportation Science 23, 288–299 (1989)
22. Hutson, V.A., ReVelle, C.S.: Indirect covering tree problems on spanning tree networks. European Journal of Operational Research 65, 20–32 (1993)
23. Melamed, I.I., Sigal, I.K.: An investigation of linear convolution of criteria in multicriteria discrete programming. Computational Mathematics and Mathematical Physics 35(8), 1009–1017 (1995)
24. Melamed, I.I., Sigal, I.K.: A computational investigation of linear parametrization of criteria in multicriteria discrete programming. Computational Mathematics and Mathematical Physics 36(10), 1341–1343 (1996)
25. Melamed, I.I., Sigal, I.K.: Numerical analysis of tricriteria tree and assignment problems. Computational Mathematics and Mathematical Physics 38(10), 1707–1714 (1998)
26. Melamed, I.I., Sigal, I.K.: Combinatorial optimization problems with two and three criteria. Doklady Mathematics 59(3), 490–493 (1999)
27. Minoux, M.: Solving combinatorial problems with combined min-max-min-sum objective and applications. Mathematical Programming 45, 361–372 (1989)
28. Perny, P., Spanjaard, O.: A preference-based approach to spanning trees and shortest paths problems. European Journal of Operational Research 162(3), 584–601 (2005)
29. Punnen, A.P.: On combined minmax-minsum optimization. Computers and Operations Research 21, 707–716 (1994)
30. Punnen, A.P., Nair, K.P.K.: A $\mathcal{O}(m \log n)$ algorithm for the max + sum spanning tree problem. European Journal of Operational Research 89, 423–426 (1996)
31. Ramos, R.M., Alonso, S., Sicilia, J., Gonzalez, C.: The problem of the optimal biobjective spanning tree. European Journal of Operational Research 111, 617–628 (1998)
32. Ruzika, S.: On Multiple Objective Combinatorial Optimization. Ph.D thesis, Technische Universität Kaiserslautern (2007)

33. Schweigert, D.: Linear extensions and efficient trees. Preprint 172, University of Kaiserslautern, Department of Mathematics (1990)
34. Schweigert, D.: Linear extensions and vector-valued spanning trees. Methods of Operations Research 60, 219–222 (1990)
35. Sergienko, I.V., Perepelitsa, V.A.: Finding the set of alternatives in discrete multicriterion problems. Kibernetika 5, 85–93 (1987)
36. Shogan, A.: Constructing a minimal-cost spanning tree subject to resource constraints and flow requirements. Networks 13, 169–190 (1983)
37. Steiner, S., Radzik, T.: Solving the biobjective minimum spanning tree problem using *k*-best algorithm. Technical Report TR-03-06, Department of Computers Science, King's College, London, UK (2003)

Engineering Route Planning Algorithms[*]

Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner

Universität Karlsruhe (TH), 76128 Karlsruhe, Germany
{delling,sanders,wagner}@ira.uka.de, mail@dominik-schultes.de

Abstract. Algorithms for route planning in transportation networks have recently undergone a rapid development, leading to methods that are up to three million times faster than Dijkstra's algorithm. We give an overview of the techniques enabling this development and point out frontiers of ongoing research on more challenging variants of the problem that include dynamically changing networks, time-dependent routing, and flexible objective functions.

1 Introduction

Computing an optimal route in a transportation network between specified source and target nodes is one of the showpieces of real-world applications of algorithmics. We frequently use this functionality when planning trips with cars or public transportation. There are also many applications like logistic planning or traffic simulation that need to solve a huge number of shortest-path queries in transportation networks. In the first part of this paper, we focus on the simplest case, a static *road* network with a fixed cost for each edge. The cost function may be any mix of travel time, distance, toll, energy consumption, scenic value, etc. associated with the edges. Some of the techniques described below work best if the cost function is positively correlated with travel time. The task is to compute the costs of optimal paths between arbitrary source-target pairs. Some preprocessing is allowed but it has to be sufficiently fast and space efficient to scale to the road network of a continent.

The main part of this paper is Section 2, which explains the ideas behind several practically successful speedup techniques for exact static routing in road networks. Section 3 makes an attempt to summarize the development of performance over time. In Section 4 we outline generalizations for public transportation, mobile devices, outputting optimal paths, and dynamically changing networks. Augmenting static techniques to time-dependent scenarios is discussed in Section 5, while Section 6 describes some experiences we made with implementing route planning algorithms for large networks. Then, Section 7 explains our experimental approach giving several examples by applying it to some algorithms we implemented. We conclude in Section 8 with a discussion of future challenges.

[*] Partially supported by DFG grants SA 933/1-3, SA 933/5-1, WA 654/16-1 and the Future and Emerging Technologies Unit of EC (IST priority – 6th FP), under contract no. FP6-021235-2 (project ARRIVAL). A preliminary version of this paper has been published in [60].

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 117–139, 2009.

2 Static Routing

We consider directed graphs $G = (V, E)$ with n nodes and $m = \Theta(n)$ edges. An edge (u, v) has the nonnegative edge weight $w(u, v)$. A shortest-path query between a source node s and a target node t asks for the minimum weight $d(s, t)$ of any path from s to t. In static routing, the edge weights do not change so that it makes sense to perform some *precomputations*, store their results, and use this information to accelerate the queries. Obviously, there is some tradeoff between query time, preprocessing time, and space for preprocessed information. In particular, for large road networks it would be prohibitive to precompute and store shortest paths between all pairs of nodes.

2.1 "Classical Results"

Dijkstra's Algorithm [20] – the classical algorithm for route planning – maintains an array of *tentative distances* $D[u] \geq d(s, u)$ for each node. The algorithm *visits* (or *settles*) the nodes of the road network in the order of their distance to the source node and maintains the invariant that $D[u] = d(s, u)$ for visited nodes. We call the rank of node u in this order its *Dijkstra rank* $\mathrm{rk}_s(u)$. When a node u is visited, its outgoing edges (u, v) are *relaxed*, i.e., $D[v]$ is set to $\min(D[v], d(s, u)+w(u, v))$. Dijkstra's algorithm terminates when the target node is visited. The size of the search space, i.e. the number of settled nodes, is $O(n)$ and $n/2$ (nodes) on the average. We will assess the quality of route planning algorithms by looking at their *speedup* compared to Dijkstra's algorithm, i.e., how many times faster they can compute shortest-path distances.

Priority Queues. A naive implementation of Dijkstra's algorithm has a running time of $O(n^2)$ since finding the next node to settle takes $O(n)$ (linear search of candidates). However, the algorithm can be implemented using $O(n)$ priority queue operations. In the comparison based model this leads to $O(n \log n)$ execution time. In other models of computation (e.g. [71]) and on the average [47], better bounds exist. However, in practice the impact of priority queues on performance for large road networks is rather limited since cache faults for accessing the graph are usually the main bottleneck. In addition, our experiments indicate that the impact of priority queue implementations diminishes with advanced speedup techniques that dramatically reduce the queue sizes.

Bidirectional Search executes Dijkstra's algorithm simultaneously forward from the source and backwards from the target. Once some node has been visited from both directions, the shortest path can be derived from the information already gathered [12]. Bidirectional search can be combined with most other speedup techniques. On the other hand, it is a necessary ingredient of many advanced techniques.

Geometric Goal Directed Search (A^*). The intuition behind goal directed search is that shortest paths 'should' lead in the general direction of the target. A^* search [32] achieves this by modifying the weight of edge (u, v) to

$w(u, v) - \pi(u) + \pi(v)$ where $\pi(v)$ is a lower bound on $d(v, t)$. Note that this manipulation shortens edges that lead towards the target. Since the added and subtracted *vertex potentials* $\pi(v)$ cancel along any path, this modification of edge weights preserves shortest paths. Moreover, as long as all edge weights remain nonnegative, Dijkstra's algorithm can still be used. The classical way to use A^* for route planning in road maps estimates $d(v, t)$ based on the Euclidean distance between v and t and the average speed of the fastest road anywhere in the network. Since this is a very conservative estimation, the speedup for finding quickest routes is rather small. Goldberg et al. [25] even report a *slow-down* of more than a factor of two since the search space is not significantly reduced but a considerable overhead is added.

2.2 Exploiting Hierarchy

Small Separators. Transportation networks are almost planar, i.e., most edges intersect only at nodes. Hence, techniques developed for planar graphs will often also work for road networks. Using $O(n \log^2 n)$ space and preprocessing time, query time $O(\sqrt{n} \log n)$ can be achieved [22,41] for directed planar graphs without negative cycles. Queries accurate within a factor $(1 + \epsilon)$ can be answered in near constant time using $O((n \log n)/\epsilon)$ space and preprocessing time [70]. Most of these theoretical approaches look difficult to use in practice since they are complicated and need superlinear space. The approach from [70] has recently been implemented and experimentally evaluated on a road network with one million nodes [52]. While the query times are very good (less than $20\,\mu$s for $\epsilon = 0.01$), the preprocessing time and space consumption are quite high (2.5 hours and 2 GB, respectively).

Multi-Level Techniques. The first published practical approach to fast route planning [67,68] uses a set of nodes V_1 whose removal partitions the graph $G = G_0$ into small components. Now consider the *overlay graph* $G_1 = (V_1, E_1)$ where edges in E_1 are *shortcuts* corresponding to shortest paths in G that do not contain nodes from V_1 in their interior. Routing can now be restricted to G_1 and the components containing s and t respectively. This process can be iterated yielding a multi-level method [69,35,36,34]. A limitation of this approach is that the graphs at higher levels become much more dense than the input graphs thus limiting the benefits gained from the hierarchy. Also, computing small separators and shortcuts can become quite costly for large graphs.

Reach-Based Routing. Let $R(v) := \max_{s,t \in V} R_{st}(v)$ denote the *reach* of node v where $R_{st}(v) := \min(d(s, v), d(v, t))$. Gutman [31] observed that a shortest-path search can be stopped at nodes with a reach too small to get to source or target from there. Variants of reach-based routing work with the reach of edges or characterize reach in terms of geometric distance rather than shortest-path distance. The first implementation had disappointing speedups (e.g. compared to [67]) and preprocessing times that would be prohibitive for large networks.

Highway Hierarchies (HHs) [58,59] group nodes and edges in a hierarchy of levels by alternating between two procedures: Contraction (i.e., node reduction) removes low degree nodes by bypassing them with newly introduced shortcut edges. In particular, all nodes of degree one and two are removed by this process. Edge reduction removes *non-highway edges*, i.e., edges that only appear on shortest paths *close* to source or target. More specifically, every node v has a neighborhood radius $r(v)$ that we are free to choose. An edge (u, v) is a highway edge if it belongs to some shortest path P from a node s to a node t such that (u, v) is neither fully contained in the neighborhood of s nor in the neighborhood of t, i.e., $d(s, v) > r(s)$ and $d(u, t) > r(t)$. In all our experiments, neighborhood radii are chosen such that each neighborhood contains a certain number H of nodes. H is a tuning parameter that can be used to control the rate at which the network shrinks. The query algorithm is very similar to bidirectional Dijkstra search with the difference that certain edges need not be expanded when the search is sufficiently far from source or target. HHs were the first speedup technique that could handle the largest available road networks giving query times measured in milliseconds. There are two main reasons for this success: Under the above contraction routines, the road network remains sparse and near planar. Furthermore, preprocessing can be done using limited local searches starting from each node which resulted in the fastest preprocessing at that time.

Advanced Reach-Based Routing. It turns out that the preprocessing techniques developed for HHs can be adapted to preprocessing reach information [26]. This makes reach computation faster and more accurate. More importantly, shortcuts make queries more effective by reducing the number of nodes traversed and by reducing the reach-values of the nodes bypassed by shortcuts. Reach-based routing is slower than HHs both with respect to preprocessing time and query time. However, the latter can be improved by a combination with goal-directed search to a point where both methods have similar performance.

Highway-Node Routing (HNR). In [65] the multi-level routing scheme with overlay graphs [67,69,35,36] is generalized so that it works with arbitrary sets of nodes rather than only with separators. This is achieved using a new query algorithm that stalls suboptimal branches of search on lower levels of the hierarchy. By using only *important* nodes for higher levels, query performance is comparable to HHs. Preprocessing is done in two phases. In the first phase, nodes are classified into levels. In the second phase, the shortcuts are recursively computed bottom up. Shortcuts from level ℓ are found by local searches in level $\ell - 1$ starting from nodes in level ℓ. This second phase is very fast and easy to update when edge weights change.

Contraction Hierarchies (CHs) are a special case of highway-node routing where we have n levels – one level for each node [23]. Such a fine hierarchy can improve query performance by a considerable factor. The queries are also further simplified since it is sufficient to search upward in a *search graph.* This property saves considerable space since each edge is only stored at its lower endpoint.

CH preprocessing proceeds in two phases. The first phase orders nodes by importance. The second phase contracts (removes) nodes in this order. When node v is contracted it is removed from the network in such a way that shortest path distances between the remaining nodes are preserved. See Fig. 1 for an example. Local searches are used to decide which of the potential shortcuts of the form $\langle u, v, w \rangle$ are needed. In a sense, contraction hierarchies are a simplification of HHs where only the node contraction phases are used (using a more careful implementation). Node ordering keeps the nodes in a priority queue with priorities based on how attractive it is to contract a node. The most important term of the priority function is the *edge difference* – how many additional edges would the graph get if a node is contracted (this value may be negative). Another important priority term ensures that nodes are removed from the networks *uniformly* rather than only in one area of the network. Due to their simplicity and efficiency, contraction hierarchies are now used in almost all of our advanced routing techniques.

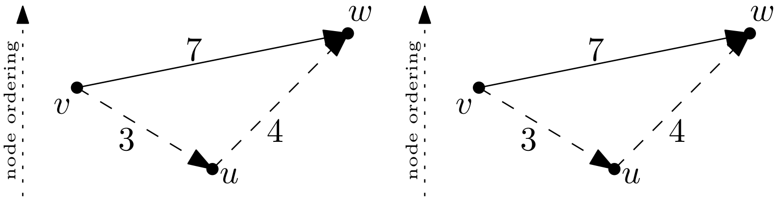


Fig. 1. Contraction with node ordering $u < v < w$. Node u is contracted by adding a shortcut from v to w and by removing the incoming and outgoing edges of u.

Distance Tables. Once a hierarchical routing technique (e.g., HH, HNR, CH) has shrunk the size of the remaining network G' to $\Theta(\sqrt{n})$, one can afford to precompute and store a complete distance table for the remaining nodes [59]. Using this table, one can stop a query when it has reached G'. To compute the shortest-path distance, it then suffices to lookup all shortest-path distances between nodes entering G' in forward and backward search respectively. Since the number of entrance nodes is not very large, one can achieve a speedup close to two compared to the underlying hierarchical technique.

Transit-Node Routing precomputes not only a distance table for important (*transit*) nodes but also all relevant connections between the remaining nodes and the transit nodes. Independently, three approaches proved successful for selecting transit nodes: separators [51,15], border nodes of a partition [3,4,2], and nodes categorized as important by other speedup techniques [3,4,61]. It turns out that for route planning in road networks, the latter approach is the most promising one. Since only about 7–10 such *access connections* are needed per node one can 'almost' reduce routing in large road networks to about 100 table lookups. Interestingly, the difficult queries are now the local ones where the shortest path does not touch any transit node. This problem can be solved by introducing several *layers* of transit nodes. Between lower layer transit nodes, only those routes need to be stored that do not touch the higher layers. Transit-node routing (e.g., using appropriate slices of a CH) reduces routing times to a few microseconds at the price of larger preprocessing time and additional space consumption.

2.3 Advanced Goal-Directed Search

Edge Labels. The idea behind edge labels is to precompute information for an edge e that specifies a set of nodes $M(e)$ with the property that $M(e)$ is a superset of all nodes that lie on a shortest path starting with e. In an s–t query, an edge e need not be relaxed if $t \notin M(e)$. In [67], $M(e)$ is specified by an *angular range*. More effective is information that can distinguish between long range and short range edges. In [74] many *geometric containers* are evaluated. Very good performance is observed for axis parallel rectangles. A disadvantage of geometric containers is that they require a complete all-pairs shortest-path computation. Faster precomputation is possible by partitioning the graph into k regions that have similar size and only a small number of boundary nodes. Now $M(e)$ is represented as a k-vector of *edge flags* [44,43,48,49,63] where flag i indicates whether there is a shortest path containing e that leads to a node in region i. Fig. 2 gives an example. Edge flags can be computed using a single-source shortest-path computation from all boundary nodes of the regions. A further improvement gets by with only one (though comparatively expensive) search for each region [33].

SHARC is an extension of the edge flag approach [6]. By using subroutines from hierarchical approaches—namely contraction—during preprocessing most of the disadvantages of edge flags can be remedied. The key observation is that it is sufficient to set suboptimal edge flags to many edges of the graph. Expensive preprocessing is then only done for important edges. In addition, SHARC extends the idea of the 2-level edge flag approach presented in [48] to a multi-level arc-flags setup. The result is a fast *unidirectional* query algorithm, which is especially advantageous in scenarios where bidirectional search is prohibitive, e.g. time-dependent networks (cf. Section 5). In general, SHARC can be interpreted as

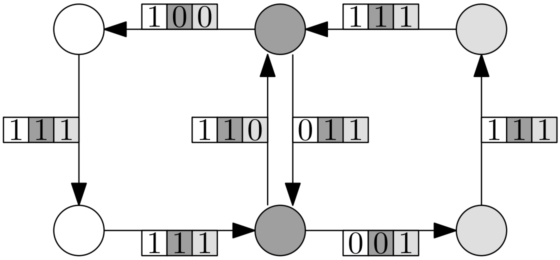


Fig. 2. Example for Arc-Flags. The graph is partitioned in 3 regions.

goal-directed technique that incorporates hierarchical aspects implicitely. Even faster query times can be achieved by a bidirectional version of SHARC.

Landmark A^* (ALT). Using the triangle inequality, quite strong bounds on shortest-path distances can be obtained by precomputing distances to a set of *landmark* nodes (≈ 20) that are well distributed over the far ends of the network [25,29]. Using reasonable space and much less preprocessing time than for edge labels, these lower bounds yield considerable speedup for route planning.

Precomputed Cluster Distances (PCD). In [46], a different way to use precomputed distances for goal-directed search is given. The network is partitioned into clusters and then a shortest connection between any pair of clusters U and V, i.e., $\min_{u \in U, v \in V} d(u, v)$, is precomputed. PCDs cannot be used together with A^* search since reduced edge weights can become negative. However, PCDs yield upper and lower bounds for distances that can be used to prune search. This gives speedup comparable to landmark-A^* using less space. Using the many-to-many routing techniques outlined in Section 4, cluster distances can also be computed efficiently.

2.4 Combinations

Bidirectional search can be profitably combined with almost all other speedup techniques. Indeed, it is a required[1] ingredient of highway hierarchies, transit-node routing, highway-node routing and contraction hierarchies and it achieves a considerable improvement for reach-based routing and edge flags. Willhalm et al. have made a systematic comparison of combinations of pre-2004 techniques [38,76,37]. Landmark A^* harmonizes very well with reach-based routing [26] whereas it gives only a small additional speedup when combined with HHs [17].

Recently, the combination of edge flags with hierarchical approaches has proved very successful [8]. Contraction hierarchies combined with edge flags yield query times almost as low as transit-node routing using less space and preprocessing time. This large reduction in preprocessing times compared to edge-flags alone stems from a restriction of the edge-flag computations to a network that is already considerably contracted. This combination is also quite effective for many types of networks where hierarchical methods alone work not as well as for road networks. Edge flags combined with transit-node routing lead to the currently fastest query times for road networks.

2.5 Differences to Commercial Systems

It turns out that commercial route planning systems and the methods described above have a lot of things in common but differ in a crucial fact. Like commercial systems, the methods from Sections 2.2 and 2.3 follow some kind of intuition a

[1] However, there are approaches that combine forward search with a type of backward graph exploration that is not shortest path search. This will become relevant in Section 5.

human uses for traveling in transportation networks. The main difference is that commercial systems might provide suboptimal results while the latter guarantee to find the shortest path. Surprisingly, settling for approximate results does not result in faster query times. In fact, the contrary is true.

For example, a common approach in commercial car navigation systems is the following: do not look at 'unimportant' streets, unless you are close to the source or target [39]. This heuristic needs careful hand tuning of road classifications to produce reasonable results but yields considerable speedups. Recalling the concept of highway hierarchies, one might notice that HH follows the same intuition: after a fixed number of hops, consider only a highway network. It turns out that one main reason for the better performance of HH compared to the heuristic is the correctness of the former. The latter has to make a precarious compromise between quality and size of the search space that relies on manual classification of the edges into levels of the hierarchy. In contrast, after setting a few quite robust tuning parameters, HH-preprocessing automatically computes a hierarchy aggressively tuned for high performance.

3 Chronological Summary – The Horse Race

Although academic research on speedup techniques for Dijkstra's algorithm started in the late nineties, motivated from timetable information [67], a boost in development was the publication of continental-sized road networks in 2005. The European network was made available for scientific use by the company PTV AG. The USA network was from publicly available geographical data [72]. Before that, input-sizes were limited and even worse, most data was confidential, e.g. the timetable data used in [67]. This lead to the problem that it was hard to compare different approaches with respect to performance. Once large road networks were available to everybody [58,19] so that speedup techniques became comparable, a "horse race" started: Which group can achieve the fastest query (and preprocessing) times on these inputs. In this Section we give a chronological summary of this race, including techniques that were published *before* 2005.

However, it is still difficult to compare speedup techniques even for road networks because there is a complex tradeoff between query time, preprocessing time and space consumption that depends on the network, on the objective function, and on the distribution of queries. Still, we believe that some ranking helps to compare the techniques. We take the liberty to speculate on the performance of some older methods that have never been run on such large graphs and whose actual implementations might fail when one would attempt it. In Tab. 1 we list speedup techniques in chronological order that are 'best' with respect to speedup for random queries and the largest networks tackled at that point. Sometimes we list variants with slower query times if they are considerably better with respect to space consumption or manageable graph size.

Before [67] the best method would have been a combination of bidirectional search with geometric A^* yielding speedups of 2–3 over unidirectional Dijkstra. The separator-based multi-level method from [67] can be expected to work

Table 1. Chronological development of the fastest speedup techniques. As date for the first publication, we usually give the submission deadline of the respective conference. If available, we always selected measurements for the European road network even if they were conducted after the first publication. Otherwise, we *linearly* extrapolated the preprocessing times to the size of Europe, which can be seen as a *lower bound.* Note that not all speedup techniques have been preprocessed on the same machine. Also note that we report the space consumption of the technique *including* the graph.

method	first	date	data	size	space	preproc.	speedup
	pub.	mm/yy	from	$n/10^6$	[B/n]	[min]	
DIJKSTRA	[20]	08/59	-	18	21	0	1
separator multi-level	[67]	04/99	[36]	0.1	?	$>$ 5 400	52
edge flags (basic)	[44]	03/04	[45]	1	35	299	523
landmark A^*	[25]	07/04	[28]	18	89	13	28
edge flags	[43,48]	01/05	[33]	18	30	1 028	3 951
HHs (basic)	[58]	04/05	[58]	18	49	161	2 645
reach + shortc. + A^*	[26]	10/05	[28]	18	100	1 625	1 559
	[28]	08/06	[28]	18	56	141	3 932
HHs + dist. tab.	[59]	04/06	[64]	18	68	13	10 364
HHs + dist. tab. + A^*	[17]	08/06	[64]	18	92	14	12 902
high-perf. multi-level	[51]	06/06	[15]	18	181	1 440	401 109
transit nodes (eco)	[3]	10/06	[64]	18	140	25	574 727
transit nodes (gen)	[3]	10/06	[64]	18	267	75	1 470 231
highway nodes	[65]	01/07	[64]	18	28	15	7 437
approx. planar $\epsilon = 0.01$	[52]	09/07	[52]	1	2 000	150	18 057
SHARC	[6]	09/07	[7]	18	34	107	21 800
bidirectional SHARC	[6]	09/07	[7]	18	41	212	97 261
contr. hier. (aggr)	[23]	01/08	[23]	18	17	32	41 051
contr. hier. (eco)	[23]	01/08	[23]	18	21	10	28 350
CH + edge flags (aggr)	[8]	01/08	[8]	18	32	99	371 882
CH + edge flags (eco)	[8]	01/08	[8]	18	20	32	143 682
transit nodes + edge flags	[8]	01/08	[8]	18	341	229	3 327 372
contr. hier. (mobile)	[62]	04/08	[62]	18	8	31	9 878

even for large graphs if implemented carefully. Computing geometric containers [67,74,75] is still infeasible for large networks. Otherwise, they would achieve much larger speedups than the separator-based multi-level method. Until recently, computing edge flags has also been too expensive for Europe and the USA but speedups beyond 523 have been observed for a graph with one million nodes [49]. Landmark A^* works well for large graphs and achieves average speedup of 28 using reasonable space and preprocessing time [25]. The implementation of HHs [58] was the first that was able to handle Europe and the USA. This implementation wins over all previous methods in almost all aspects. A combination of reach-based routing with landmark A^* [26] achieved better query times for the USA at the price of a considerably higher preprocessing time. At first, that code did not work well on the European network because it is difficult to handle the present long-distance ferry connections, but later it could be considerably improved [27]. By introducing distance tables and numerous other improvements, highway hierarchies took back the lead in query time [59] at the same time using an order of magnitude less preprocessing time than [58]. The cycle of innovation accelerated even further in 2006. Müller [51] aggressively precomputes the pieces of the search space needed for separator-based multi-level routing. At massive expense of space and preprocessing time, this method can achieve speedups around 400 000. (The original implementation cannot directly measure this because it has large overheads for disk access and parsing of XML-data). Independently, transit-node routing was developed [3], that lifts the speedup to six orders of magnitude and completely replaces Dijkstra-like search by table lookups. This approach was further accelerated by combining transit nodes with edge flags [8], yielding speedups of over 3 millions.

However, the story was not over because speedup techniques have to fulfill several properties, in addition to speed, if they should be used in real-world scenarios: Memory consumption should be as low as possible, simple updating of preprocessing due to traffic jams should be possible, and speedup techniques should work in time-dependent networks. Highway-Node Routing [65] and contraction hierarchies [23,62] fulfill the first two requirements, while SHARC [6,14] was developed for the latter scenario. These scenarios are the topic of Sections 4 and 5.

4 Generalizations

Many-to-Many Routing. In several applications we need complete distance tables between specified sets of source nodes S and target nodes T. For example, in logistics optimization, traffic simulation, and also within preprocessing techniques [46,3]. Many non-goal-directed bidirectional search methods [67,26,65]) can be adapted in such a way that only a single forward search from each source node and a single backward search from each target node is needed [42]. The basic idea is quite simple: Store the backward search spaces. Arrange them so that each node v stores an array of pairs of the form $(t, d(v, t))$ for all target nodes that have v in their backward search space. When a forward search from s settles a node v, these pairs are scanned and used to update the tentative distance from s to t. This is very efficient because the intersection between any two forward and backward search spaces is small and because scanning an array is much faster than priority queue operations and edge relaxations governing the cost of Dijkstra's algorithm. For example, for $|S| = |T| =$10 000, the implementation in [23] needs only about 10s.

Outputting Paths. The efficiency of many speedup techniques stems from introducing shortcut edges and distance table entries that replace entire paths in the original graph [67,58,26,59,51,3]. A disadvantage of these approaches is that the search will output only a 'summary description' of the optimal path that involves shortcuts. Fortunately, it is quite straightforward to augment the shortcuts with information for unpacking them [17,42,3,23,62]. Since one can afford to precompute unpacked representations of the most frequently needed long-distance shortcuts, outputting the path turns out to be up to four times *faster* than just traversing the edges in the original graph.

Mobile Route Planning. Most of the techniques described above can be adapted to mobile devices such as car navigation systems or cell phones. However, space is at a premium here and most of the data has to reside on an external memory device such as flash memory that has high access latencies and where write accesses are very expensive. Therefore, the first measure is to keep the priority queue and the nontrivial tentative distances in the fast memory. Goldberg and Werneck [29] successfully implemented the ALT algorithm on a Pocket PC. Their largest road network (North America, 29 883 886 nodes) occupies 3 735 MB and a random query takes 329 s. The RE algorithm [31,27] has been implemented on a mobile device, yielding query times of "a few seconds including path computation and search animation" and requiring "2–3 GB for USA/Europe" [24]. Contraction hierarchies have been implemented using careful blocking of nodes that are often accessed together and using aggressive variable-bitlength encoding for edges and edge weights [62]. This implementation needs only 140 MByte space for the European network and 69 ms query time on a 330 MHz ARM 11 processor without any preloaded information in the fast memory.

Turn Penalties. On most road crossings, going straight takes less time than, e.g., turning left. Such turn penalties (and disallowed turns) can be modelled using *edge based routing* where road segments become nodes of the graph and edges connect possible pairs of successive road segments. The disadvantage of this model is that a naive implementation takes several times more space than node-based routing. This problem can be circumvented by running logical edge-based routing based on bidirectional Dijkstra on a node-based physical representation. A thin interface layer makes an on-the-fly conversion. For a small subset of important nodes in the edge-based graph, an explicit contraction hierarchy is computed. The implicit Dijkstra search can switch to a fast contraction-hierarchy query when the search is covered by important nodes. Thus, most advantages of both approaches are combined [50,73].

Flexible Objective Functions. The objective function in road networks depends in a complex way on the vehicle (fast, slow, too heavy for certain bridges, etc.) the behavior and goals of the driver (cost sensitive, thinks he is fast, etc.), the load, and many other aspects. While the appropriate edge weights can be computed from a few basic parameters, it is not feasible to perform preprocessing for all conceivable combinations. Currently, our best answer to this problem is highway-node routing/contraction hierarchies [65,23]. Assuming that the important nodes are important for any reasonable objective function, only the second phase of preprocessing needs to be repeated. This is an order of magnitude faster than computing a HH or the node ordering of contraction hierarchies.

Dynamization. In online car navigation, we want to take traffic jams etc. into account. At first glance, this is fatal for most speedup techniques since even a single traffic jam can invalidate any of the precomputed information. However, we can try to selectively update only the information affected by the traffic jam and/or relevant to the queries at hand. Updating the preprocessing of Geometric Containers has been analyzed in [75]. By only allowing increases in the sizes of containers, queries stay correct, but performance may be worse than recomputing the containers from scratch. Landmark A^* can be dynamized either by noticing that lower bounds remain valid when edge weights can only increase, or by using known dynamic graph algorithms for updating the shortest-path trees from the landmarks [18]. Highway-node routing was originally developed for dynamization [65]. Indeed, it allows fast and easy updates (2–40 ms per changed edge weight depending on the importance of the edge). Even faster dynamization is possible by leaving the preprocessed information unchanged and only causing the query algorithm to descend the hierarchy when it encounters unreliable information. This approach scales to quite large numbers of *delays* using an iterative approach that only takes edges into account that actually affect the current query [64]. In the meantime this approach has been successfully adapted to mobile contraction hierarchies, generalizing [62].

Multi-Criteria Routing. The fastest route in transportation networks is often not the "best" one. For example, users traveling by train may be willing to accept longer travel times if the number of required transfers is lower or the cost of a journey with longer duration is cheaper. We end up in a multi-criteria scenario [53,57,21] in which none of the high-performance approaches developed in the last years can be applied easily. The adaption of a fast method to this scenario is one of the main challenges in the near future.

Multimodal Routing. Often, we use more than one transportation network for traveling. For example, we first use our car to get to the train station, use the train to get to the airport and finally board a plane. Planning a route in such a combined network has to fulfill certain constraints, e.g., your own car is only available at the beginning of the journey or we want to avoid switching the type of transportation too frequentely. These constraints can be modeled by adding appropriate labels to each edge [1,34]. Unfortunately, using a fast routing algorithm in such a *label-constrained* scenario is very complicated and hence, another challenging task.

5 Time-Dependency

As already mentioned, most developed techniques require the network to be *static* or only allow a small number of updates [65,18]. In practice, however, travel duration often depends on the departure time. It turns out that efficient models for routing in almost all transportation systems, e.g., timetable information for railways or scheduling for airplanes, are based on *time-dependent* networks. Moreover, road networks are not static either: there is a growing body of data on travel times of important road segments stemming from road-side sensors, GPS systems inside cars, traffic simulations, etc. Using this data, we can assign *speed profiles* to roads. This yields a time-dependent road network.

Switching from a static to a time-dependent scenario is more challenging than one might expect: The input size increases drastically as travel times on congested motorways change during the day. On the technical side, most static techniques rely on bidirectional search, i.e., a second search is started from the target. This concept is prohibited in time-dependent scenarios as the arrival time would have to be known in advance for such a procedure. Moreover, possible problem statements for shortest paths become even more complex in such networks. A user could ask at what time she should depart in order to spend as little time traveling as possible. As a result, none of the existing high-performance techniques can be adapted to this realistic scenario easily.

5.1 Modeling Issues

The major difference between static and time-dependent routing is the usage of functions instead of constants for specifying edge weights. We use piece-wise linear functions for modeling time-dependency in road networks[2]. Each edge gets assigned a number of sample points that depict the travel time on this road at the specific time. Evaluating the travel time for an edge at time τ is then done by linear interpolation between the points left and right to τ. In order to allow a polynomial time exact solution [40,56], we assume that the network fulfills the *FIFO property* or *non-overtaking property*: if A leaves an arbitrary node s before B, B cannot arrive at any node t before A.

Currently, research concentrates on solving the earliest arrival problem in time-dependent networks, i.e., find the quickest route from s to t for given time of departure τ. Note that this problem is closely related to the latest departure problem, i.e., find the quickest route from s to t such that you arrive at t at a given time τ' [13].

Rolling-Out Time-Dependency. Another approach to model time-dependency is to roll out the time component. In the *time-expanded* approach for timetable information [66,54], each time-dependent edge is multiplied such that each edge represents exactly one connection and a timestamp is assigned to each node. Although the resulting graphs are time-independent, the main downside of this approach is the increase in input size. While this is still practicable for timetable information [53,30], the time-expanded graphs get way too big for road networks. In addition, adaption of speedup techniques to the time-expanded approach is more complicated than expected [9]. Hence, the time-dependent approach seems more promising.

5.2 Basic Algorithmic Toolbox

Analyzing all speedup techniques, preprocessing relies on two major ingredients: (local) DIJKSTRA-searches and contraction. Hence, for correct routing in time-dependent networks, both ingredients need to be augmented.

Generalizing Dijkstra's Algorithm. A straightforward extension [11] of DIJKSTRA's algorithm is capable of computing the distance between s and t

[2] Note that this is flexible enough to accurately integrate time-tabled connections such as ferries.

when departing from s at time τ. However, we may also need to compute a *profile*, i.e. the distance between s and t for *all* departure times. It turns out that this is more expensive but can be computed by a label-correcting variant of DIJKSTRA's algorithm [13]: distance labels now become functions of time and edge relaxations compute minima of two time-dependent functions; nodes may have to be scanned several times when parts of the their distance label improve. An interesting result from [13] is the fact that the runtime of label-correcting algorithms highly depends on the complexity of the edge-functions.

Contraction. The second ingredient of high-performance speedup techniques is contraction (cf. Section 2). Basically, we may leave the principle untouched: unimportant nodes are removed from the graph and shortcuts are added in order to preserve distances between remaining nodes. However, contraction in time-dependent road networks is more expensive, in terms of space consumption, than in time-independent networks. Let $P(f)$ be the number of interpolation points of the function f assigned to an edge (u, v). Then the composed function $f \oplus g$, modeling the duration for traversing g after f, may have up to $P(f)+P(g)$ interpolation points in the worst case. This is one of the main problems when routing in time-dependent graphs: Almost all speedup techniques developed for static scenarios rely on adding long shortcuts to the graph. While this is "cheap" for static scenarios, the insertion of time-dependent shortcuts yields a high amount of preprocessed data.

An interesting observation is that in timetable networks, the problem of increasing interpolation does not exist. More precisely, $P(f \oplus g) = \min\{P(f), P(g)\}$ holds as the number of relevant departure times is dominated by the edge with less connections.

5.3 Adapting Speedup Techniques

Up to now, three different approaches have successfully been adapted to the time-dependent scenario. They either use a unidirectional search or perform a backward search that limits the node set the forward search has to visit.

A* with landmarks (ALT) was the first natural choice for adaption to time-dependent scenarios. Like for the dynamic time-independent scenarios, ALT performs correct queries as long as potentials are feasible. So, by using the lower bound of each edge during preprocessing, ALT-queries stay correct. Unfortunately, a unidirectional variant [18] only leads to a mild speedups of 3–5. A speedup of 30 can be obtained if we are willing to accept suboptimal paths being up to 15% longer than the shortest [55]. This can be achieved by a bidirectional approach. Here, a time-independent backward search bounds the node set that has to be examined by the forward search. This approach can be enhanced by performing ALT only on a small core built by contraction during preprocessing [16].

SHARC. A disadvantage of ALT is its rather small speedup even in static scenarios. This was the main motivation for the development of SHARC, a unidirectional speedup technique [6] being as fast as bidirectional approaches. It is based on two foundations: contraction and edge flag computation. An edge flag is now set as soon as it is important for at least one departure time [14]. As a result, quickest paths for all departure times between two points have arc-flags set to true. A straight-forward approach is to compute flags using the label correcting algorithm described above. While this works for large timetable graphs, preprocessing takes way too long for road networks. By setting suboptimal arc-flags, preprocessing times can be reduced but for the price of query performance. Depending on the degree of perturbation, time-dependent SHARC is up to 214 times faster than Dijkstra.

Contraction Hierarchies [5]. Due to their simplicity and good performance, contraction hierarchies are an interesting candidate for generalization. In the most simple implementation, the first preprocessing phase (node ordering) can be done on the static graph. Only the second phase needs to replace ordinary Dijkstra searches by profile searches. This is straight-forward in principle but challenging because a naive implementation is very expensive. The query algorithm is slightly less obvious. The simultaneous forward and backward Dijkstra-searches of the static version are replaced by four activities[3]: 1. A time dependent forward-upward Dijkstra-search. 2. Backward *exploration* of the downward edges leading to the target. Note that in most other hierarchical search techniques this weakening of the backward search would imply traversing the entire graph whereas the DAG (directed acyclic graph) property of contraction hierarchies makes it possible that only a small subset of the nodes is visited. 3. Pruning of supoptimal paths (when forward distance plus a lower bound on the distance to the target exceed some upper bound). 4. A forward-downward Dijkstra search starting simultaneously from all the promising places where forward search and backward exploration met that is confined to the edges explored in component 2. Various forms of upper and lower bounds on travel time can be used to improve pruning, accelerate preprocessing, and save space.

6 Implementation

Advanced algorithms for routing in road networks require thousands of lines of well written code and hence require considerable programming skill. In particular, it is not trivial to make the codes work for large networks. Here is an incomplete list of problems and complications that we have seen in routing projects: Graphs have to be glued together from several files. Tools for reading files crash for large graphs. Algorithm library code cannot handle large graphs at all. The code slows down several times when switching from a custom graph representation to an algorithm library. 32-bit code will not work. Libraries do not work with 64-bit code. Our conclusion from these experiences was to design our own graph data structures adapted to the problem at hand. We use C++

[3] We avoid the term ‘phase’ here since the activities can be interspersed in various ways.

with encapsulated abstract data types. Templates and inline functions make this possible without performance penalties.

Speedup techniques developed by algorithmicists usually come with high level arguments why they should yield optimal paths. While this is already much more robust than purely heuristic algorithms, we sometimes observed that subtle details only revealed by a detailed correctness proof can yield suboptimal paths. For example, we had several cases where the algorithm considered was only correct when all shortest paths are unique.

There are plenty of things that can go wrong both with the algorithms and their implementations. The implementation can help here with extensive consistency checks in assertions and experiments that are always checked against naive implementations, i.e., queries are checked against Dijkstra's algorithm and fast preprocessing algorithms are checked against naive or old implementations. On the long run one also needs a flexible visualization tool that can draw pieces of large graphs, paths, search spaces, and node sets. Since we could not find tools for this purpose that scale to large road networks, we implemented our own system [10].

7 Methodology of Experiments

Evaluating speedup techniques on large networks is not as trivial as one might expect. For example, an obvious choice of queries uses randomly selected node pairs on the largest available graph. Although the resulting average query time is a meaningful number, it is not quite satisfactory since most queries will produce very long paths (thousands of kilometers) that are actually rare in practice. One possible solution is to use random queries on a variety of subgraphs. However, this leads to a plethora of arbitrary choices that make it difficult to compare results. In particular, authors will be tempted to choose only those subgraphs for which their method performs well.

Sets of real world queries would certainly be interesting, but it is unlikely that a sample taken from one server is actually representative for the entire spectrum of route planning applications. We therefore chose a more systematic approach [58] that has also been adopted in several other studies: We generate a random query with a specified 'locality' r by choosing a random starting node s, and a target node t with Dijkstra rank $\mathrm{rk}_s(t) = r$ (i.e., the r-th node visited by a Dijkstra search from s). In our studies, we generate many such queries for each r which is a power of two. We then plot the distribution with median, quartiles, and outliers for each of these values of r. For the European road network, Fig. 3 shows the results for contraction hierarchies, CHs combined with edge flags, and for transit-node routing. We view it as quite important to give information on the entire distribution since some speedup techniques have large fluctuations in query time.

In some cases, e.g., for contraction hierarchies, it is also possible to compute good upper bounds on the search space size of *all* queries that can ever happen for a given graph [59,64,23]. Figure 4 gives the upper bounds for contraction

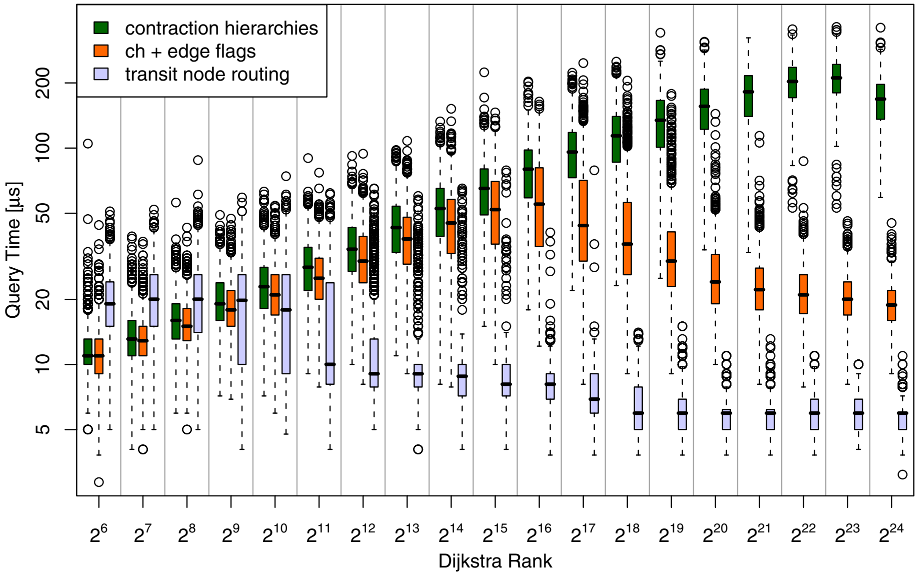


Fig. 3. Query performance of various speedup techniques against Dijkstra rank. We observe that adding edge flags to contraction hierarchies accelerates long-range queries, while transit node routing also accelerates mid-range queries (compared to pure contraction hierarchies).

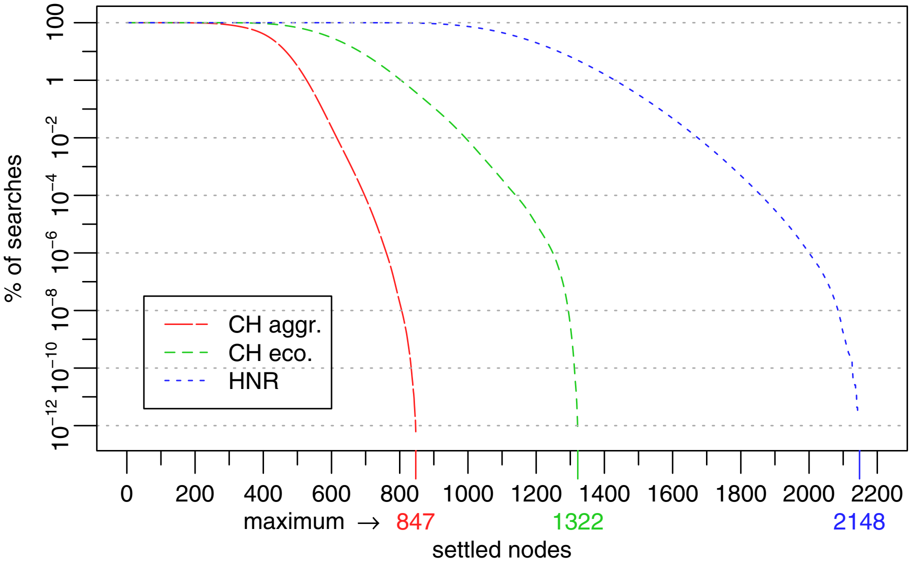


Fig. 4. Worst case upper bounds for various hierarchical speedup techniques

hierarchies and highway-node routing. We view this as a quite good surrogate for the absence of meaningful worst case upper bounds that would apply to all conceivable networks.

8 Conclusions and Open Problems

Speedup techniques for routing in static road networks have made tremendous progress in the last few years. While for challenging applications such as logistics planning and traffic simulation, query times cannot be fast enough, for other applications the query times are more than satisfying: other overheads like displaying routes or transmitting them over the network are the bottleneck once the query time is below a few milliseconds.

A major challenge is to close the gap to theory, e.g., by giving meaningful characterizations of 'well-behaved' networks that allow provably good worst-case bounds. In particular, we would like to know for which networks the existing techniques will also work, e.g., for communication networks, VLSI design, social networks, computer games, graphs derived from geometric routing problems, ...

Perhaps the main academic challenge is to go beyond static point-to-point routing. Although first techniques already provide promising results, the gap between static and time-dependent routing is still very big. A very important topic for the furture is to reduce preprocessing space of time-dependent techniques. Here, the main problem lies in the mentioned problem that shortcuts are "cheap" in static scenarios, while in time-dependent scenarios shortcuts can become highly complex objects. Further beyond that, we want multi-criteria optimization for individual paths and we want to run realistic traffic simulations.

Acknowledgements

We would like to thank our coauthors on route planning, Holger Bast, G. Veit Batz, Reinhard Bauer, Chris Barrett, Keith Bisset, Stefan Funke, Robert Geisberger, Martin Holzer, Ekkehard Köhler, Goran Konjevod, Sebastian Knopp, Leo Liberti, Madhav V. Marathe, Domagoj Matijevic, Jens Maue, Rolf Möhring, Kirill Müller, Matthias Müller-Hannemann, Giacomo Nannicini, Evangelia Pyrga, Dennis Schieferdecker, Heiko Schilling, Frank Schulz, Christian Vetter, Lars Volker, Karten Weihe, Thomas Willhalm, and Christos Zaroliagis for their valuable contributions. We also had many interesting discussions with Rob van den Berg, Sebastian Egner, Andrew Goldberg, Joaquim Gromicho, Stefan Hug, Ali Nowbakht Irani, Riko Jacob, Ulrich Lauther, Ramon Lentink, Dennis Luxen, Paul Perdon, Martin Pfeifle, Mikkel Thorup, Jaques Verrier, Peter Vortisch, and Renato Werneck. Finally, we thank PTV AG and HaCon for providing us with real-world data for scientific use.

References

1. Barrett, C., Bisset, K., Holzer, M., Konjevod, G., Marathe, M.V., Wagner, D.: Engineering Label-Constrained Shortest-Path Algorithms. In: Fleischer, R., Xu, J. (eds.) AAIM 2008. LNCS, vol. 5034, pp. 27–37. Springer, Heidelberg (2008)
2. Bast, H., Funke, S., Matijevic, D.: TRANSIT Ultrafast Shortest-Path Queries with Linear-Time Preprocessing. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear) (accepted for publication)
3. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In Transit to Constant Shortest-Path Queries in Road Networks. In: Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX 2007), pp. 46–59. SIAM, Philadelphia (2007)
4. Bast, H., Funke, S., Sanders, P., Schultes, D.: Fast Routing in Road Networks with Transit Nodes. Science 316(5824), 566 (2007)
5. Batz, G.V., Geisberger, R., Sanders, P.: Time Dependent Contraction Hierarchies - Basic Algorithmic Ideas. Technical report, ITI Sanders, Faculty of Informatics, Universität Karlsruhe (TH) (2008), arXiv:0804.3947v1 [cs.DS]
6. Bauer, R., Delling, D.: SHARC: Fast and Robust Unidirectional Routing. In: Munro, I., Wagner, D. (eds.) Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX 2008), pp. 13–26. SIAM, Philadelphia (2008)
7. Bauer, R., Delling, D.: SHARC: Fast and Robust Unidirectional Routing. Submitted to the ACM Journal of Experimental Algorithmics (2008) (full paper)
8. Bauer, R., Delling, D., Sanders, P., Schieferdecker, D., Schultes, D., Wagner, D.: Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 303–318. Springer, Heidelberg (2008)
9. Bauer, R., Delling, D., Wagner, D.: Experimental Study on Speed-Up Techniques for Timetable Information Systems. In: Liebchen, C., Ahuja, R.K., Mesa, J.A. (eds.) Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2007), pp. 209–225. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2007)
10. Bingmann, T.: Visualisierung sehr großer Graphen. Student Research Project (2006)
11. Cooke, K., Halsey, E.: The Shortest Route Through a Network with Time-Dependent Intermodal Transit Times. Journal of Mathematical Analysis and Applications (14), 493–498 (1966)
12. Dantzig, G.B.: Linear Programming and Extensions. Princeton University Press, Princeton (1962)
13. Dean, B.C.: Continuous-Time Dynamic Shortest Path Algorithms. Master's thesis, Massachusetts Institute of Technology (1999)
14. Delling, D.: Time-Dependent SHARC-Routing. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 332–343. Springer, Heidelberg (2008)
15. Delling, D., Holzer, M., Müller, K., Schulz, F., Wagner, D.: High-Performance Multi-Level Routing. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear) (accepted for publication)

16. Delling, D., Nannicini, G.: Bidirectional Core-Based Routing in Dynamic Time-Dependent Road Networks. In: Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC 2008). LNCS. Springer, Heidelberg (2008) (to appear)
17. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway Hierarchies Star. In: Demetrescu, et al. (eds.) [19]
18. Delling, D., Wagner, D.: Landmark-Based Routing in Dynamic Graphs. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 52–65. Springer, Heidelberg (2007)
19. Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.): 9th DIMACS Implementation Challenge - Shortest Paths (November 2006)
20. Dijkstra, E.W.: A Note on Two Problems in Connexion with Graphs. Numerische Mathematik 1, 269–271 (1959)
21. Disser, Y., Müller-Hannemann, M., Schnee, M.: Multi-Criteria Shortest Paths in Time-Dependent Train Networks. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 347–361. Springer, Heidelberg (2008)
22. Fakcharoenphol, J., Rao, S.: Planar Graphs, Negative Weight Edges, Shortest Paths, and near Linear Time. Journal of Computer and System Sciences 72(5), 868–889 (2006)
23. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 319–333. Springer, Heidelberg (2008)
24. Goldberg, A.: Personal communication (2008)
25. Goldberg, A.V., Harrelson, C.: Computing the Shortest Path: A* Search Meets Graph Theory. In: Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2005), pp. 156–165 (2005)
26. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Reach for A*: Efficient Point-to-Point Shortest Path Algorithms. In: Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX 2006), pp. 129–143. SIAM, Philadelphia (2006)
27. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Better Landmarks Within Reach. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 38–51. Springer, Heidelberg (2007)
28. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Reach for A*: Shortest Path Algorithms with Preprocessing. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear) (accepted for publication)
29. Goldberg, A.V., Werneck, R.F.: Computing Point-to-Point Shortest Paths from External Memory. In: Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX 2005), pp. 26–40. SIAM, Philadelphia (2005)
30. Gunkel, T., Müller-Hannemann, M., Schnee, M.: Improved Search for Night Train Connections. In: Liebchen, C., Ahuja, R.K., Mesa, J.A. (eds.) Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2007), pp. 243–258. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2007)
31. Gutman, R.J.: Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In: Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX 2004), pp. 100–111. SIAM, Philadelphia (2004)

32. Hart, P.E., Nilsson, N., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics 4, 100–107 (1968)
33. Hilger, M., Köhler, E., Möhring, R.H., Schilling, H.: Fast Point-to-Point Shortest Path Computations with Arc-Flags. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear)
34. Holzer, M.: Engineering Planar-Separator and Shortest-Path Algorithms. Ph.D thesis, Universität Karlsruhe (TH), Fakultät für Informatik (2008)
35. Holzer, M., Schulz, F., Wagner, D.: Engineering Multi-Level Overlay Graphs for Shortest-Path Queries. In: Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX 2006). SIAM, Philadelphia (2006)
36. Holzer, M., Schulz, F., Wagner, D.: Engineering Multi-Level Overlay Graphs for Shortest-Path Queries. ACM Journal of Experimental Algorithmics (2008) (to appear)
37. Holzer, M., Schulz, F., Wagner, D., Willhalm, T.: Combining Speed-up Techniques for Shortest-Path Computations. ACM Journal of Experimental Algorithmics 10 (2006)
38. Holzer, M., Schulz, F., Willhalm, T.: Combining Speed-up Techniques for Shortest-Path Computations. In: Ribeiro, C.C., Martins, S.L. (eds.) WEA 2004. LNCS, vol. 3059, pp. 269–284. Springer, Heidelberg (2004)
39. Ishikawa, K., Ogawa, M., Azuma, M., Ito, T.: Map Navigation Software of the Electro-Multivision of the 91 Toyoto Soarer. In: Proceedings of the Vehicle Navigation and Information Systems Conference (VNIS 1991), pp. 463–473. IEEE Computer Society, Los Alamitos (1991)
40. Kaufman, D.E., Smith, R.L.: Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application. Journal of Intelligent Transportation Systems 1(1), 1–11 (1993)
41. Klein, P.N.: Multiple-Source Shortest Paths in Planar Graphs. In: Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2005), pp. 146–155 (2005)
42. Knopp, S., Sanders, P., Schultes, D., Schulz, F., Wagner, D.: Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In: Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX 2007), pp. 36–45. SIAM, Philadelphia (2007)
43. Köhler, E., Möhring, R.H., Schilling, H.: Acceleration of Shortest Path and Constrained Shortest Path Computation. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 126–138. Springer, Heidelberg (2005)
44. Lauther, U.: An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In: Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung, vol. 22, pp. 219–230. IfGI prints (2004)
45. Lauther, U.: An Experimental Evaluation of Point-To-Point Shortest Path Calculation on Roadnetworks with Precalculated Edge-Flags. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear)
46. Maue, J., Sanders, P., Matijevic, D.: Goal Directed Shortest Path Queries Using Precomputed Cluster Distances. In: Àlvarez, C., Serna, M. (eds.) WEA 2006. LNCS, vol. 4007, pp. 316–327. Springer, Heidelberg (2006)

47. Meyer, U.: Single-Source Shortest-Paths on Arbitrary Directed Graphs in Linear Average-Case Time. In: Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2001), pp. 797–806 (2001)
48. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning Graphs to Speed Up Dijkstra's Algorithm. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 189–202. Springer, Heidelberg (2005)
49. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning Graphs to Speedup Dijkstra's Algorithm. ACM Journal of Experimental Algorithmics 11, 2.8 (2006)
50. Müller, K.: Berechnung kürzester Pfade unter Beachtung von Abbiegeverboten. Student Research Project (2005)
51. Müller, K.: Design and Implementation of an Efficient Hierarchical Speed-up Technique for Computation of Exact Shortest Paths in Graphs. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik (June 2006)
52. Muller, L.F., Zachariasen, M.: Fast and Compact Oracles for Approximate Distances in Planar Graphs. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 657–668. Springer, Heidelberg (2007)
53. Müller-Hannemann, M., Schnee, M.: Finding All Attractive Train Connections by Multi-Criteria Pareto Search. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 246–263. Springer, Heidelberg (2007)
54. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.: Timetable Information: Models and Algorithms. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 67–90. Springer, Heidelberg (2007)
55. Nannicini, G., Delling, D., Liberti, L., Schultes, D.: Bidirectional A* Search for Time-Dependent Fast Paths. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 334–346. Springer, Heidelberg (2008)
56. Orda, A., Rom, R.: Shortest-Path and Minimum Delay Algorithms in Networks with Time-Dependent Edge-Length. Journal of the ACM 37(3), 607–625 (1990)
57. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient Models for Timetable Information in Public Transportation Systems. ACM Journal of Experimental Algorithmics 12, Article 2.4 (2007)
58. Sanders, P., Schultes, D.: Highway Hierarchies Hasten Exact Shortest Path Queries. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 568–579. Springer, Heidelberg (2005)
59. Sanders, P., Schultes, D.: Engineering Highway Hierarchies. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 804–816. Springer, Heidelberg (2006)
60. Sanders, P., Schultes, D.: Engineering Fast Route Planning Algorithms. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 23–36. Springer, Heidelberg (2007)
61. Sanders, P., Schultes, D.: Robust, Almost Constant Time Shortest-Path Queries in Road Networks. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) Shortest Paths: Ninth DIMACS Implementation Challenge, DIMACS Book. American Mathematical Society, Providence (2008) (to appear) (accepted for publication)
62. Sanders, P., Schultes, D., Vetter, C.: Mobile Route Planning. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 732–743. Springer, Heidelberg (2008)
63. Schilling, H.: Route Assignment Problems in Large Networks. Ph.D thesis, Technische Universität Berlin (2006)
64. Schultes, D.: Route Planning in Road Networks. Ph.D thesis, Universität Karlsruhe (TH), Fakultät für Informatik (February 2008)

65. Schultes, D., Sanders, P.: Dynamic Highway-Node Routing. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 66–79. Springer, Heidelberg (2007)
66. Schulz, F.: Timetable Information and Shortest Paths. Ph.D thesis, Universität Karlsruhe (TH), Fakultät für Informatik (2005)
67. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. In: Vitter, J.S., Zaroliagis, C.D. (eds.) WAE 1999. LNCS, vol. 1668, pp. 110–123. Springer, Heidelberg (1999)
68. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. ACM Journal of Experimental Algorithmics 5 (2000)
69. Schulz, F., Wagner, D., Zaroliagis, C.: Using Multi-Level Graphs for Timetable Information in Railway Systems. In: Mount, D.M., Stein, C. (eds.) ALENEX 2002. LNCS, vol. 2409, pp. 43–59. Springer, Heidelberg (2002)
70. Thorup, M.: Compact Oracles for Reachability and Approximate Distances in Planar Digraphs. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2001), pp. 242–251. IEEE Computer Society Press, Los Alamitos (2001)
71. Thorup, M.: Integer Priority Queues with Decrease Key in Constant Time and the Single Source Shortest Paths Problem. In: Proceedings of the 35th Annual ACM Symposium on the Theory of Computing (STOC 2003), June 2003, pp. 149–158 (2003)
72. U.S. Census Bureau, Washington, DC. UA Census 2000 TIGER/Line Files (2002), http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html
73. Volker, L.: Route planning in road networks with turn costs. Studienarbeit, Universität Karlsruhe, Institut für theoretische Informatik (2008)
74. Wagner, D., Willhalm, T.: Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 776–787. Springer, Heidelberg (2003)
75. Wagner, D., Willhalm, T., Zaroliagis, C.: Geometric Containers for Efficient Shortest-Path Computation. ACM Journal of Experimental Algorithmics 10, 1.3 (2005)
76. Willhalm, T.: Engineering Shortest Paths and Layout Algorithms for Large Graphs. Ph.D thesis, Universität Karlsruhe (TH), Fakultät für Informatik (2005)

From State-of-the-Art Static Fleet Assignment to Flexible Stochastic Planning of the Future*

Sven Grothklags[1], Ulf Lorenz[2], and Burkhard Monien[3]

[1] Lufthansa Systems Berlin GmbH
Airline Planning & Control
Salzufer 8, D-10587 Berlin

[2] Darmstadt University of Technology
Institute of Mathematics
Schlossgartenstr. 7, D-64289 Darmstadt

[3] University of Paderborn, Faculty of Computer Science,
Electrical Engineering and Mathematics
Fürstenallee 11, D-33102 Paderborn

Abstract. Given a flight schedule, which is a set of non-stop flights, called legs, with specified departure and arrival times, and a set of aircraft types, called subfleets, the fleet assignment problem is to determine which aircraft type should fly each leg. The objective is to maximize the overall profit.

Usually, planners assume that precise input data are determininstically available at planning time. As a consequence, an important insufficiency of modern industrial plans, especially flight-plans, is their lack of robustness. Disruptions prevent from operating as planned before and induce high costs for trouble shooting.

One reason for the uncertainties in the data for fleet assignment is that this problem is part of a long row of optimization problems of an airline. Therefore, important restrictions of later steps like connection dependent ground times should be considered in the fleet assignment problem. We show how connection dependent ground times can be added to the fleet assignment problem and presents three optimization methods, varying in run time and solution quality, that can solve real-world problem instances with more than 6000 legs within minutes.

Moreover, real or believed non-determinism leads to inevitable uncertainties in input data. As a consequence, instead of a traditional plan, a flexible strategy which reacts on different relizations of the uncertain data is demanded. The Repair Game is a formalization of a planning task, and playing it performs disruption management and generates robust plans with the help of game tree search. We introduce the game and present experimental results of a feasibility study.

1 Introduction

For operating an airline, several optimization problems have to be solved. These include, e.g., network planning, aircraft and crew scheduling. In this paper we address

* This work has been partially supported by the European Union within the 6th Framework Program under contract 001907 (DELIS) and the German Science Foundation (DFG), SFB 614 (Selbstoptimierende Systeme des Maschinenbaus) and SPP 1126 (Algorithmik großer und komplexer Netzwerke).

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 140–165, 2009.

the fleet assignment problem (FAP), and how connection dependent ground times can be considered. Moreover, an important problem in state-of-the-art aircraft planning is to react with an instant decision, after a certain disruption hinders the company to act as planned before. Therefore, we show how stochastic uncertainties can be incorporated into the FAP within the frame of games against Nature. The problem belongs to stochastic multi-stage optimization problems.

In the following, we introduce the fleet-assignment problem, characterize the most influencing research fields for our work here, and outline our approach for disruption management.

1.1 The Deterministic Fleet-Assignment Problem (FAP)

Briefly, for the FAP a flight schedule is given, consisting of a set of legs (flights without stopover) with departure and arrival station (airport), departure time and block/ground time for every subfleet (aircraft type). A subfleet has to be assigned to every leg while maximizing the profit and not exceeding the given number of aircraft per subfleet.

The assigned subfleet type has great influence on a leg's profit due to the subfleet's seating capacity, fuel consumption, airport fees and personnel cost. So choosing the ”right” one is of vital importance for an airline. But the FAP is only one, though important, optimization problem in a long row and the following stages use the solution of the FAP as input. Therefore the main operational constraints of the following stages should be adhered by the FAP as far as possible, in order to produce solutions that are valid and useful for the following optimization problems, especially the aircraft rotation building and tail assignment.

One operational restriction that arises in these later stages is considering minimum turn times of aircraft that depend on the arriving *and* departing leg. We call such turn times *connection dependent ground times*. A common situation, where connection dependent ground times are needed, occurs at stations that have distinct terminals for domestic and international flights. If an aircraft arriving with a domestic flight wants to succeed with an international flight it must be towed to a different terminal, which can last more than an hour. But if it proceeds with another domestic flight it can stay at the terminal and the minimum ground time is much shorter.

We present three methods, one MIP based and two local search based approaches, to solve the FAP with connection dependent ground times. The MIP approach combines ideas of two well-known MIP models for the FAP to get an improved model that is able to solve real-world problem instances with connection dependent ground times in reasonable times [1,16]. The local search approach is an extension of a previously developed heuristic that is part of a commercial airline tool of our industrial partner Lufthansa Systems.

Because operating aircraft is one of the largest expense factors in airlines, the FAP is of considerable importance to airlines and has attracted researchers for many years, theoretically [15,18,26] and practically [1,6,8,16,30,35]. A common approach for solving the FAP is to model the problem as a mixed integer program (MIP) like in [16]. But there also exist a number of heuristic approaches [6,14,33,34,36], mainly based on local search. Recently, many researchers started to extend the basic FAP by adding additional restrictions [5,7] or incorporation additional optimization problems [4,33].

1.2 Fleetassignment under Uncertainty

Multistage Decisions under Risk. Despite all successes of deterministic optimization, we observe inconsistencies between planned and real operations. These inconsistencies come to our mind in form of disruptions, and the reason for disruptions in the operations obviously stems from the fact that planners lack information about the real behavior of the environment at planning time: Traditionally, plans are built which maximize profits over 'expected' or just estimated input data, but it is also convenient to assume that it is more realistic to optimize the expected payoff over all possible scenarios instead. We assume that data are only approximately known and that they are modeled with the help of distributions. E.g., in the airline example, we know a distribution over a leg's (i.e. flight's) possible arrival times. Moreover, it can be assumed that we learn about realizations of random variables in the course of time. This view leads us to the field of 'multistage decisions under risk', related to linear stochastic programming [10,28], stochastic Optimization [21], game playing [3], re-planning [20] and others [31].

Game Tree Search. A multistage decision process as described above can be interpreted as a game between Nature and an operator. We have to find a decision and one time step later we learn about realizations of some of the random variables and we have to take a further decision etc. If we assume all variables being discrete, the resulting structure forms out a so called game tree. The link between stochastic optimization and game tree search is established due to complexity theory: In many settings, stochastic optimization problems, as well as the games traditionally played with the help of game tree search, are both PSPACE-complete [25].

Game tree search itself is the core of most attempts to make computers play games. The game tree acts as an error filter and examining the tree behaves similar to an approximation procedure.

At some level of branching, the complete game tree (as defined by the rules of the game) is cut, the artificial leaves of the resulting subtree are evaluated with the help of heuristics, and these values are propagated to the root [19,27] of the game tree as if they were real ones. For 2–person zero-sum games, computing this heuristic minimax value is by far the most successful approach in computer games history, and when Shannon [32] proposed a design for a chess program in 1949 it seemed quite reasonable that deeper searches lead to better results. Indeed, the important observation over the last 40 years in the chess game and some other games is: *the game tree acts as an error filter*. Therefore, the faster and the more sophisticated the search algorithm, the better the search results! This, however, is not self-evident, as some theoretical analyzes show [2,23,17].

New Approach. Our approach [9] can roughly be described by inspecting a (stochastic) planning task in a 'tree-wise' manner. Let a tree T be given that represents the possible scenarios as well as our possible actions in the forecast time-funnel. It consists of two different kinds of nodes, MAX nodes and AVG nodes. A node can be seen as a 'system state' at a certain point of time at which several alternative actions can be performed/scenarios can happen. Outgoing edges from MAX nodes represent our possible actions, outgoing edges from AVG nodes represent the ability of Nature to act in various ways. Every path from the root to a leaf can then be seen as a possible solution of our planning task; our actions are defined by the edges we take at MAX nodes under the condition that Natures acts as described by the edges that lead out of AVG nodes.

The leaf values are supposed to be known and represent the total costs of the 'planning path' from the root to the leaf. The value of an inner MAX node is computed by taking the maximum of the values of its successors. The value of an inner AVG node is built by computing a weighted average of the values of its successor nodes. The weights correspond to realization probabilities of the scenarios.

Let a so called *max-strategy* S be a subtree of T which contains the root of T, and which contains exactly one successor at MAX nodes, and all successors that are in T at AVG nodes. Each strategy S shall have a value $f(S)$, defined as the value of S's root. A *principle variation* $p(S)$, also called *plan*, of such a max-strategy can be determined by taking the edges of S leaving the MAX nodes and a highest weighted outgoing edge of each AVG node. The connected path that contains the root is $p(S)$. We are interested in the plan $p(S_b)$ of the best strategy S_b and in the expected costs $E(S_b)$ of S_b. The expected costs $E(p)$ of a plan p are defined as the expected costs of the best strategy S belonging to plan p, e.g. $E(p) = max\{E(S) \mid p(S) = p\}$.

This model might be directly applied in some areas, as e.g. job shop scheduling [22], not, however, in applications which are sensible to temporary plan deviations. If a job shop scheduling can be led back to the original plan, the changes will nothing cost, as the makespan will stay as it was before.

This is different in airline fleet assignments. Because differences between planned operations and real operations cause costs, the expected costs associated with a given plan are not the same before and after the plan is distributed to customers. A plan gets a value of its own once it is published. Thus, it is possible to find back to the original plan after some while, but nevertheless, costs occur. Tree nodes will have to be to identified with a pair of the system state plus the path, how the state has been reached.

1.3 Organization of This Paper

In Section 2, the problem definitions, models and algorithms in the context of the deterministic FAP are described. The section itself is divided into several subsections. In subsection 2.1 the FAP with connection dependent ground times is defined as a mathematical model. Subsection 2.3 introduces a new MIP model for the FAP and in subsection 2.4 a local search based heuristic is presented. In subsection 2.5 a generalized preprocessing technique is described that reduces the problem size of an FAP instance.

We introduce the Repair Game in section 3 as a reasonable formalization of the airline planning task on the level of disruption fighting. Section 3 describes the Repair Game, its formal definition (subsection 3.1), as well as an interpretation of the definition and an example in subsection 3.2. In subsection 3.3 we describe a prototype, which produces robust repair decisions for disrupted airline schedules, on the basis of the Repair Game.

In the 4^{th} section experimental results are presented. In subsection 4.1, we present the experiments for improvements in the context of the deterministic FAP, describing the influence of different preprocessing techniques and comparing the run times of FAPs with and without connection dependent ground times. In subsection 4.2, we compare the results of our new approach under uncertainty with an optimal repair procedure (in the traditional sense). A comparison of the sequential and a parallel version of our prototype is additionally given.

The paper ends with concluding remarks in section 5.

2 Problem Definitions, Models and Algorithms for the Deterministic FAP

2.1 FAP Problem Definition

The FAP occurs in different variations during the planning process of an airline. In the midterm strategic planning the FAP is solved for a *typical cyclic* time period. An often suggested interval length is one day which applies to many large U.S. domestic airlines. The FAP for a more irregular schedule is solved on a weekly basis like schedules of European or internationally operating airlines.

During short-term tactical planning the FAP must be solved for a concrete fully-dated time interval of up to six weeks. Most airlines take the typical period of the strategic phase, roll it out to some larger time interval, include additional flights (e.g., for holidays) and remove canceled flights. Here the interval is not cyclic but you may have to consider a configuration of aircraft waiting at specific stations at the start and/or end of the interval. The non-cyclic FAP can also be used during operations for rescheduling when you have to deal with heavy disruptions of the (planned) schedule. Here you need very fast FAP solvers but you only have to deal with small time intervals of one to three days.

Input Data and Notations. For simplicity and ease of notation we restrict ourselves to the non-cyclic FAP for the rest of the paper. Nevertheless note that all presented models, algorithms and preprocessing techniques can be easily adopted for cyclic FAPs. An instance of an FAP consists of the following input data:

$\mathcal{F}$	set of available subfleets
N_f	number of aircraft available for subfleet $f \in \mathcal{F}$
$\mathcal{L}$	set of legs (schedule)
$\mathcal{F}_l$	set of subfleets that can be assigned to leg $l \in \mathcal{L}$, $\mathcal{F}_l \subseteq \mathcal{F}$
$p_{l,f}$	profit of leg $l \in \mathcal{L}$ when flown by subfleet $f \in \mathcal{F}_l$
s_l^{dep}	departure station (airport) of leg $l \in \mathcal{L}$
s_l^{arr}	arrival station of leg $l \in \mathcal{L}$
$t_{l,f}^{dep}$	departure time of leg $l \in \mathcal{L}$ when flown by subfleet $f \in \mathcal{F}_l$
$t_{l,f}^{arr}$	arrival time of leg $l \in \mathcal{L}$ when flown by subfleet $f \in \mathcal{F}_l$
$g(k,l,f)$	minimum ground time needed at station s_k^{arr} when an aircraft of subfleet f operates legs k and l successively; $s_k^{arr} = s_l^{dep}$ must hold

The only difference of the FAP with connection dependent ground times compared to the classical FAP is the minimum ground time function $g(k,l,f)$, which depends on the arriving *and* departing leg. It is defined as a function, because the minimum ground time normally is computed out of some attributes of the legs k and l (e.g. station, international or domestic flight, ...) and the subfleet f. A complete table of all possibilities would be to large and hardly maintainable. In the classical FAP the minimum ground time only depends on the arriving flight k and the subfleet f and can therefore directly be incorporated into the arrival time $t_{k,f}^{arr}$.

Connection Network. The FAP with connection dependent ground times described here can be defined by a well-known IP called connection network [1]. The IP consists of $|\mathcal{F}|$ flow networks, one for each subfleet. The legs are the nodes of the network and arcs represent possible connections between legs. We define the set of valid successors $C_{l,f}$ (predecessors $C_{l,f}^{-1}$) of a leg l when flown by subfleet f:

$$C_{l,f} = \left\{k \in \mathcal{L} | f \in \mathcal{F}_k, s_l^{arr} = s_k^{dep}, t_{l,f}^{arr} + g(l,k,f) \leq t_{k,f}^{dep}\right\} \cup \{*\}$$

$$C_{l,f}^{-1} = \left\{k \in \mathcal{L} | f \in \mathcal{F}_k, s_k^{arr} = s_l^{dep}, t_{k,f}^{arr} + g(k,l,f) \leq t_{l,f}^{dep}\right\} \cup \{*\}$$

$C_{l,f}$ contains all possible legs, that an aircraft of subfleet f can succeed with *after* operating leg l. Therefore a leg $k \in C_{l,f}$ must be compatible with subfleet f, must depart at the arrival station of leg l and the ground time between the arrival time of leg l and the departure time of leg k must not be less than the minimum ground time of the legs l and k. The additional element $*$ is used if the aircraft does not fly any further leg until the end of the planning period. $C_{l,f}^{-1}$ contains all possible legs, that an aircraft can fly *before* operating leg l and $*$ means that l is the first leg of the aircraft.

The connection network IP uses binary variables $x_{k,l,f}$. A variable $x_{k,l,f}$ is one iff legs k and l are flown successively by an aircraft of subfleet f. The variable $x_{*,l,f}$ ($x_{l,*,f}$) is used for "connections" without predecessor (successor), that is l is the first (last) leg flown by an aircraft of subfleet f during the planning period. With these notations we can write the FAP as follows:

$$\text{maximize} \qquad \sum_{k \in \mathcal{L}} \sum_{f \in \mathcal{F}_k} p_{k,f} \left(\sum_{l \in C_{k,f}} x_{k,l,f} \right) \tag{1}$$

$$\text{subject to} \qquad \sum_{f \in \mathcal{F}_k} \sum_{l \in C_{k,f}} x_{k,l,f} = 1 \qquad \forall k \in \mathcal{L} \tag{2}$$

$$\sum_{k \in C_{l,f}^{-1}} x_{k,l,f} - \sum_{m \in C_{l,f}} x_{l,m,f} = 0 \qquad \forall l \in \mathcal{L}, f \in \mathcal{F}_l \tag{3}$$

$$\sum_{l \in \mathcal{L}} x_{*,l,f} \leq N_f \qquad \forall f \in \mathcal{F} \tag{4}$$

$$x_{k,l,f} \in \{0,1\} \qquad \forall k \in \mathcal{L}, f \in \mathcal{F}_k, l \in C_{k,f} \tag{5}$$

Condition (2) and (5) ensure that every leg is flown by exactly one subfleet. Equation (3) models the flow conservation and condition (4) limits the number of aircraft of each subfleet by limiting the *total inflow* of each subfleet. The objective function (1) maximizes the total profit.

The connection network is in principle able to solve the FAP with connection dependent ground times but has one huge disadvantage that makes it impractical for real-world instances: the model requires a big number of binary variables that grows quadratically with the number of legs. Even when using sophisticated preprocessing techniques only small FAPs (a few hundred legs) can be solved by this model. So there is an urgent need for faster solution methods.

In the following, we present a new MIP model for the FAP with connection dependent ground times. The new model is a hybrid of the connection network introduced in section 2.1 and the so called time space network [16]. The time space network is probably the most popular method to solve the FAP but it cannot handle connection dependent ground times.

2.2 Time Space Network

As we already mentioned the time space network cannot model connection dependent ground times. Ground times may only depend on the arriving leg l and the subfleet f. Therefore this *leg dependent* ground time can be added to the arrival time of the leg $t^{arr}_{l,f}$ and does not need to be considered further. *Every* leg with a departure time greater equal $t^{arr}_{l,f}$ is a valid successor.

As the connection network, the time space network also consists of $|\mathcal{F}|$ flow networks, one for each subfleet. But the individual flow networks are constructed differently: The nodes in the network are the flight events on the stations (possible arrivals and departures of aircraft). The edges of the network are comprised of flight arcs and ground arcs. A flight arc connects a departure and an arrival event of one leg flown by a specific subfleet. The ground arcs connect two subsequent flight events on one station. A flight event can be identified by a triple (t, s, f), where t is the time of arrival (or departure), s is the station of arrival (or departure) and f is the subfleet the event belongs to.

We need the following notations to define the time space network:

$\mathcal{V}$ set of all flight events
$v^{dep}_{l,f} = (t^{dep}_{l,f}, s^{dep}_{l}, f)$; flight event that corresponds to the departure of leg l when flown by subfleet f
$v^{arr}_{l,f} = (t^{arr}_{l,f}, s^{arr}_{l}, f)$; flight event that corresponds to the arrival of leg l when flown by subfleet f
v^{+} subsequent flight event of $v \in \mathcal{V}$ on the same station of the same subfleet; $*$ if v is the last flight event
v^{-} preceding flight event of $v \in \mathcal{V}$ on the same station of the same subfleet; $*$ if v is the first flight event
$\mathcal{V}^{*}_{f}$ set of flight events of subfleet f that have no predecessor

The time space network MIP uses binary variables $y_{l,f}$ for the flow on flight arcs. $y_{l,f}$ is one iff leg l is flown by subfleet f. Moreover we use non-negative variables z_{v,v^+} to represent the flow on the ground arcs. With these notations we can write:

$$\text{maximize} \qquad \sum_{l\in\mathcal{L}} \sum_{f\in\mathcal{F}_l} p_{l,f} y_{l,f} \tag{6}$$

$$\text{subject to} \qquad \sum_{f\in\mathcal{F}_l} y_{l,f} = 1 \qquad \forall l \in \mathcal{L} \tag{7}$$

$$\sum_{v^{arr}_{l,f}=v} y_{l,f} - \sum_{v^{dep}_{l,f}=v} y_{l,f} + z_{v^-,v} - z_{v,v^+} = 0 \qquad \forall v \in \mathcal{V} \tag{8}$$

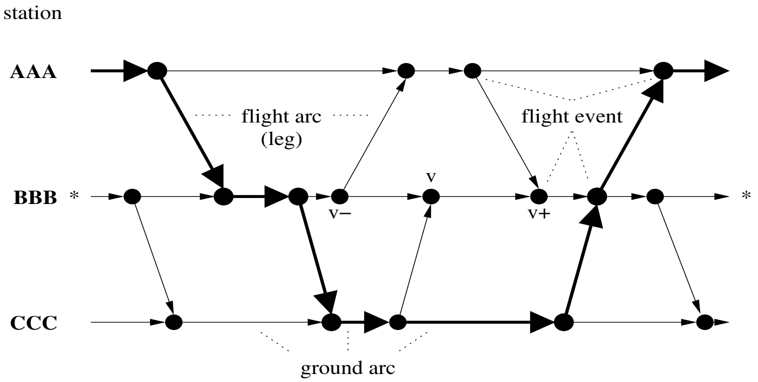


Fig. 1. Time Space Network

$$\sum_{v \in \mathcal{V}_f^*} z_{*,v} \leq N_f \qquad \forall f \in \mathcal{F} \tag{9}$$

$$y_{l,f} \in \{0,1\} \qquad \forall l \in \mathcal{L}, f \in \mathcal{F}_l \tag{10}$$

$$z_{v,v^+} \geq 0 \qquad \forall v \in \mathcal{V} \tag{11}$$

$$z_{*,v} \geq 0 \qquad \forall v \in \mathcal{V}^* \tag{12}$$

Condition (7) and (10) ensure that every leg is flown by exactly one subfleet. Equality (8) models the flow conservation for each flight event. Condition (9) limits the number of aircraft of each subfleet and conditions (11) and (12) ensure that the flow on the ground arcs is non-negative. The objective function (6) maximizes the total profit. Figure 1 shows an example.

Note that in contrast to the connection network the number of binary variables only grows linearly with the number of legs and that the number of conditions is about the same[1]. Moreover, it is sufficient to choose the z-variables non-integer. This is because an indeed possible fractional value of a z-variable can only be transported to another z-variable, due to equation (8). The fractional part of a z-variable can therefore, and because equation (9) stays valid and because the z-variables are not part of the objective, be substracted from the corresponding airport without changing the solution.

2.3 New MIP Model

Our new MIP model for the FAP with connection dependent ground times can be seen as a hybrid of the time space network of section 2.2 and the connection network of section 2.1. What makes the time space network unsuitable for dealing with connection dependent ground times is the fact that it allows *all* connections (k, l) to be established if $t^{arr}_{k,f} \leq t^{dep}_{l,f}$ holds. The main idea for out new model is to increase $t^{arr}_{k,f}$ in such a way that *all* later departing legs form valid connections. By doing so we may miss some

[1] The number of flight events is bounded by $2 \cdot |\mathcal{L}| \cdot |\mathcal{F}|$ and can be reduced below $|\mathcal{L}| \cdot |\mathcal{F}|$ by a simple preprocessing step described in [16].

valid successors of leg k that depart earlier than the adjusted arrival time of leg k and these connections are handled explicitly like in the connection network.

In practice the differences of connection dependent ground times for an arriving leg hardly exceeds one or two hours. Therefore the arrival time of a leg also needs to be increased only by these one or two hours at most, so you can expect that the number of *missed valid* connections is quite small and you only need a small number of additional binary variables for these connections in our new model compared to the time space network.

The adjusted arrival time of a leg l when flown by subfleet f can be computed by

$$t'_{l,f} = \max\left\{t^{dep}_{k,f} + 1 | f \in \mathcal{F}_k, s^{arr}_l = s^{dep}_k, t^{dep}_{k,f} < t^{arr}_{l,f} + g(l,k,f)\right\} \cup \left\{t^{arr}_{l,f}\right\}$$

The expression simply determines the latest "compatible" departing leg which does not form a valid connection with leg l and sets the adjusted arrival time $t'_{l,f}$ of leg l one time unit above. If no such invalid departing leg exists the arrival time is left unchanged. Knowing the adjusted arrival time we can define the set of "missed" valid successors $C_{l,f}$ (predecessors $C^{-1}_{l,f}$):

$$C_{l,f} = \left\{k \in \mathcal{L} | f \in \mathcal{F}_k, s^{arr}_l = s^{dep}_k, t^{arr}_{l,f} + g(l,k,f) \leq t^{dep}_{k,f}, t^{dep}_{k,f} < t'_{l,f}\right\}$$

$$C^{-1}_{l,f} = \left\{k \in \mathcal{L} | f \in \mathcal{F}_k, s^{arr}_k = s^{dep}_l, t^{arr}_{k,f} + g(k,l,f) \leq t^{dep}_{l,f}, t^{dep}_{l,f} < t'_{k,f}\right\}$$

There is only one difference in the definition of $C_{l,f}$ and $C^{-1}_{l,f}$ compared to the definition in the section 2.1 of the connection network: We are only interested in legs that depart during the interval $[t^{arr}_{l,f}, t'_{l,f})$.

As the connection and time space network our new MIP model consists of $|\mathcal{F}|$ flow networks. Each flow network consists of two different kind of nodes: leg nodes that correspond to the nodes of the connection network and event nodes that correspond to the nodes of the time space network. The edges of the network are comprised of the flight and ground arcs of the time space network and the connection arcs of the connection network. Like in the time space network ground arcs connect successive event nodes of one station. The flight arcs still connect the departure event of a leg with its arrival event but they are divided in two parts and in between they visit the corresponding leg node. Finally the connection arcs establish the "missed" valid connections between leg nodes.

Besides the changed definition of $C_{l,f}$ and $C^{-1}_{l,f}$ we also redefine the meaning of the arrival event $v^{arr}_{l,f}$ to be the triple $(t'_{l,f}, s^{arr}_l, f)$, that is we use the adjusted arrival time here. Our new MIP model uses binary variables $x_{k,l,f}$, $y^{dep}_{l,f}$ and $y^{arr}_{l,f}$. $x_{k,l,f}$ directly corresponds to the x-variables in the connection network. As mentioned earlier the flight arcs of the time space network are divided in two parts and $y^{dep}_{l,f}$ ($y^{arr}_{l,f}$) is the flight arc that connects the departure event $v^{dep}_{l,f}$ (arrival event $v^{arr}_{l,f}$) with the corresponding leg node. Finally we use non-negative variables z_{v,v^+} as in the time space network to represent the flow on the ground arcs. With these notations our new MIP model can be defined as follows:

$$\text{maximize} \sum_{k \in \mathcal{L}} \sum_{f \in \mathcal{F}_k} p_{k,f} \left(y^{arr}_{k,f} + \sum_{l \in C_{k,f}} x_{k,l,f} \right) \tag{13}$$

$$\text{subject to} \qquad \sum_{f \in \mathcal{F}_k} \left(y^{arr}_{k,f} + \sum_{l \in C_{k,f}} x_{k,l,f} \right) = 1 \qquad \forall k \in \mathcal{L} \tag{14}$$

$$y^{dep}_{l,f} + \sum_{k \in C^{-1}_{l,f}} x_{k,l,f} - y^{arr}_{l,f} - \sum_{m \in C_{l,f}} x_{l,m,f} = 0 \qquad \forall l \in \mathcal{L}, f \in \mathcal{F}_l \tag{15}$$

$$\sum_{v^{arr}_{l,f} = v} y^{arr}_{l,f} - \sum_{v^{dep}_{l,f} = v} y^{dep}_{l,f} + z_{v^-,v} - z_{v,v^+} = 0 \qquad \forall v \in \mathcal{V} \tag{16}$$

$$\sum_{v \in \mathcal{V}^*_f} z_{*,v} \leq N_f \qquad \forall f \in \mathcal{F} \tag{17}$$

$$x_{k,l,f} \in \{0,1\} \qquad \forall\ k \in \mathcal{L}, f \in \mathcal{F}_k, l \in C_{k,f} \tag{18}$$

$$y^{dep}_{l,f}, y^{arr}_{l,f} \in \{0,1\} \qquad \forall\ l \in \mathcal{L}, f \in \mathcal{F}_l \tag{19}$$

$$z_{v,v^+} \geq 0 \qquad \forall\ v \in \mathcal{V} \tag{20}$$

$$z_{*,v} \geq 0 \qquad \forall\ v \in \mathcal{V}^* \tag{21}$$

Whether a leg k is flown by subfleet f or not can be identified by the outflow[2] of the corresponding leg node, that is $y^{arr}_{k,f} + \sum_{l \in C_{k,f}} x_{k,l,f}$. Therefore the objective function (13) maximizes the total profit. Equation (14) ensures that every leg is operated by exactly one subfleet. Equation (15) ensures the flow conservation condition for leg nodes and equation (16) for event nodes. Condition (17) limits the number of used aircraft per subfleet. Conditions (18)–(21) define the domains of the used variables.

Compared to the time space network we double the number of y-variables and add some binary x-variables to the model. Furthermore we introduce the flow equations (15) for the leg nodes, that is we roughly double the number of equations needed. As mentioned earlier the number of "missed" valid connections (and therefore x-variables) is quite low in practice and there are also many legs, for which $C_{l,f}$ (or $C^{-1}_{l,f}$) is empty. For these legs you can substitute $y^{arr}_{l,f}$ for $y^{dep}_{l,f} + \sum_{k \in C^{-1}_{l,f}} x_{k,l,f}$ ($y^{dep}_{l,f}$ for $y^{arr}_{l,f} + \sum_{m \in C_{l,f}} x_{l,m,f}$) and remove the corresponding leg node equation from the model[3]. So normally we will end up with a model that is not much larger than a corresponding time space network MIP. Figure 2 shows an example.

2.4 Local Search Heuristics

We enhanced our previously developed local search based heuristics [14] to be able to deal with connection dependent ground times. The specialized neighborhood can

[2] Or, alternatively, inflow.

[3] The model will collapse into a normal time space network if all sets $C_{l,f}$ and $C^{-1}_{l,f}$ are empty. This is the case for FAPs *without* connection dependent ground times.

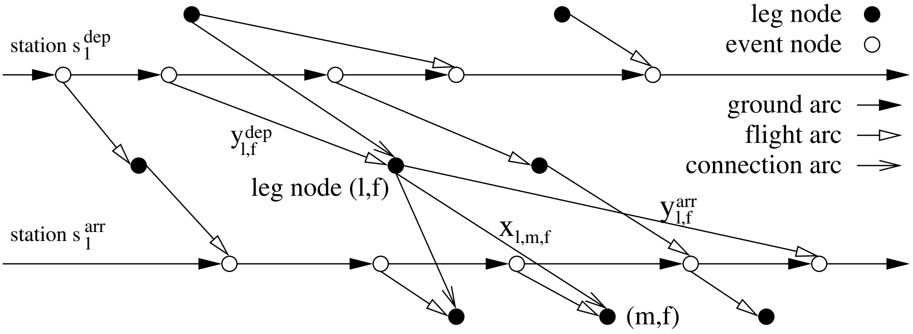


Fig. 2. Example for a hybrid of time space network and connection network

be used in a hill climbing or simulated annealing framework to produce high quality solutions in short times. The simulated annealing heuristic uses an adaptive cooling schedule described in [24].

Since flight schedules evolve over time, solving the FAP does not have to start from scratch. Changes to a flight schedule are integrated into the old schedule by airline experts, so that for the real-world fleet assignment instances an initial solution is given which can then be used by our local search algorithms.

So clearly, the most challenging part for using Local Search to solve the FAP is how to define the neighborhood. Therefore we will restrict to this topic for the rest of the section. We allow two transitions which we call *change* and *swap*. In a nutshell, change and swap look for leg sequences that do not use more aircraft than the current solution when moved to a different subfleet. Although it might be possible to find other solutions by violating temporarily the number of available aircraft of a subfleet, we found it very difficult to reach from such an invalid solution a valid solution, since valid neighbors are rare.

We initially developed our neighborhood for the FAP without considering connection dependent ground times. The following paragraphs shortly present the basic ideas of the original neighborhood. Moreover, we describe the necessary extensions to be able to deal with connection dependent ground times.

Waiting Function. Crucial for the efficient computation of our neighborhood transitions is the waiting function ([13], or Fig. 3). Given a solution to the FAP the waiting function $W_{s,f}(t)$ counts the number of aircraft of subfleet f available on station s at time t, such that $\forall t : W_{s,f}(t) \geq 0$ and $\exists t' : W_{s,f}(t') = 0$. An island of $W_{s,f}(t)$ is an interval (t_1, t_2) where $W_{s,f}(t)$ is strictly positive $\forall t \in (t_1, t_2)$ and $W_{s,f}(t_1) = 0 = W_{s,f}(t_2)$[4]. Every flight event (departure or arrival) of a given solution belongs exactly to one of these islands.

The waiting functions can be used in a number of ways. The value $W_{s,f}(-\infty)$ at the beginning of the planning interval tells you the number of aircraft of subfleet f that must be available at station s at the beginning of the planning interval to be able to operate the schedule. These values can be used to determine the number of aircraft needed by

[4] $t_1 = -\infty$ ($t_2 = \infty$) is allowed. In such a case $W_{s,f}(-\infty)$ ($W_{s,f}(\infty)$) does not need to be zero.

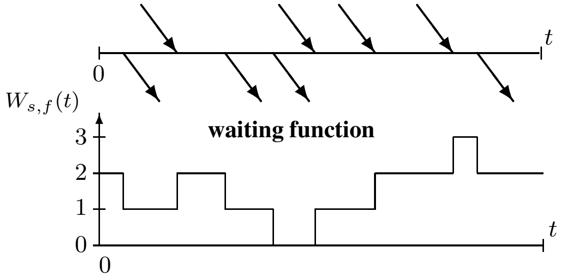


Fig. 3. Example of a waiting function with two islands for an airport s and a subfleet f. The upper part shows the distribution of landings (upper arrows) and the distribution of starts (lower arrows).

a solution. Furthermore it is known that all connections between arriving and departing legs must lie *within* the islands of the waiting function and, more importantly, that it is always *possible* to build these connections within the islands. And finally, the waiting function is an important tool to efficiently construct leg sequences for our change and swap transitions that do not increase the number of used aircraft.

Change and Swap. We call $(l_0, \ldots, l_n)$ a *leg sequence*, if the arrival airport of l_i is the departure airport of leg l_{i+1} for all $i = 0, \ldots, k-1$ and all legs $l_0, \ldots, l_n$ are assigned to the same subfleet. Note that a leg sequence is not (necessarily) representing a part of an aircraft rotation, i.e. a successor l_{i+1} can depart earlier than its predecessor l_i. Leg sequences are generated by a depth first search with limited degree at the search nodes. The possible successors in a leg sequence are not allowed to increase the number of aircraft used when the leg sequence is changed or swapped to another subfleet. For the computation of a successor the islands of the waiting function are used. The exact generation process of leg sequences is quite complex. In this paper we can only give a sketch of the two transitions defining the neighborhood and using the leg sequences as building blocks.

The Change. The change transition alters the assigned subfleet for a leg sequence which starts and ends at the same airport A. On the input of a randomly chosen leg l with f being the subfleet currently assigned to l and a subfleet g the change generates a leg sequence S_f that starts with leg l and to which subfleet g can be assigned without using more aircraft than the current solution. For this, during the whole time of S_f subfleet g has to provide an aircraft waiting on the ground on A; therefore l and the last leg of S_f must lie within one island of $W_{A,g}$.

The Swap. The swap transition exchanges the assigned subfleets f and g among two leg sequences S_f and S_g. To maintain the current distribution of aircraft at the beginning and end of the planning interval the two leg sequences have to start at the same airport A and end at the same airport B[5]. Let f be the subfleet currently assigned to a randomly

[5] If you are allowed to alter the distribution of aircraft this rule can be modified appropriately, but the details are quite complex and omitted in this paper. The same holds for the change transition.

chosen leg l. Then, on input of leg l and a subfleet g the swap transition computes first a leg sequence S'_f starting with l whose legs are currently assigned to subfleet f and that can also be flown by subfleet g. The depth first search construction of leg sequence S_g also starts at station A and searches for a compatible leg sequence of legs flown by subfleet g that ends at the same station as a *subsequence* S_f of S'_f. Then S_f and S_g form a valid swap transition.

Extensions for Connection dependent Ground Times. Our original neighborhood for the FAP has the same difficulties with connection dependent ground times as the time space network. By using the waiting functions we can assure that a solution of the FAP (without connection dependent ground times) does not exceed the available number of aircraft but we do not know the concrete connections between arriving and departing legs. We only know that there *exists* at least one valid set of connections and that these connections must lie within the islands of the waiting function. But this is only true if *all* legs, that depart later than an arriving leg l within an island, are valid successors of l, and this does not (always) hold if you have to consider connection dependent ground times.

Because the waiting function has proven to be a fast and powerful tool to construct transitions for the FAP, we adhered to this concept for the FAP with connection dependent ground times and extended it by additionally storing a *valid* successor for each leg. The generation procedure of leg sequences still uses the waiting function as major tool, but it is modified to ensure that after a transition there still is a valid successor for each leg.

The computation of successors can be solved independently for each station s and subfleet f. This can be done by calculating a complete matching on the bipartite graph of arriving and departing legs of subfleet f on station s. The nodes of this bipartite graph are the arriving and departing legs and an arriving leg k is connected with a departing leg l iff they can form a valid connection: $t^{arr}_{k,f} + g(k,l,f) \leq t^{dep}_{l,f}$. The waiting function further helps to reduce the size of the bipartite graphs, for which matchings have to be computed. As all connections must lie within islands, it is sufficient to calculate the matchings for each island separately.

The matchings are calculated by a well-known maximum-flow algorithm for the maximum matching problem for bipartite graphs. Note that these matchings only have

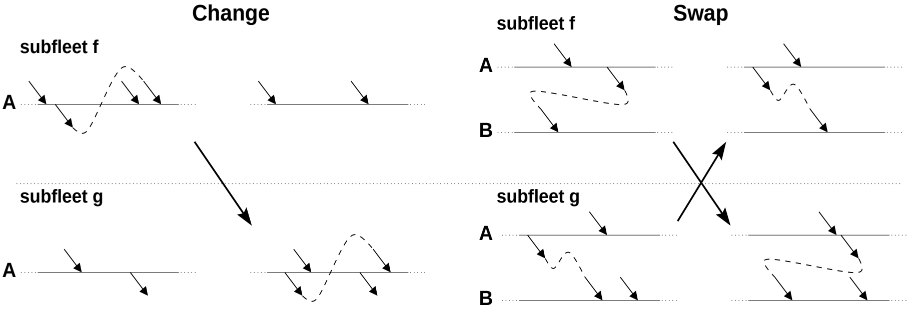


Fig. 4. Neighborhood operations: change and swap. Arcs represent incoming and outgoing legs on a station.

to be computed from scratch once, namely at the beginning of the local search algorithm. Afterwards you only have to deal with small updates to the islands, when single legs are added or removed. These updates happen obviously when you *switch* to a neighbor of the current solution but they also occur during the construction of leg sequences to *test* whether or not a leg sequence can be moved to another subfleet without destroying the complete matchings. In both cases, we only need to perform very few (one or two) augmentation steps of the maximum-flow algorithm to restore the complete matching or to discover that no complete matching exists anymore[6].

There are two simple preprocessing techniques that enhance the performance of our extended neighborhood considerably. To ensure that the waiting function harmonizes well with the added matching algorithm the arrival time $t^{arr}_{l,f}$ of a leg l when flown by subfleet f should be increased until the *first valid* departing leg is encountered, e.g. set it to $t'_{l,f} = \min\left\{t^{dep}_{k,f} | t^{arr}_{l,f} + g(l,k,f) \leq t^{dep}_{k,f}\right\} \cup \{\infty\}$ and adjust $g(l,\cdot,f)$ to not loose valid successors. Thereby the waiting function implicitly forbids connections for all departing legs in the interval $[t^{arr}_{l,f}, t'_{l,f})$.

Because the maintenance of complete matchings for the waiting functions is quite expensive, it should only be done for waiting functions that actually need these additional information to ensure that there always exist valid connections for all legs. If for *all* legs l that arrive at a station $s = s^{arr}_l$ and that are compatible with a subfleet f holds, that *all* later departing legs are valid successors, then the waiting function $W_{s,f}$ is enough to ensure that a complete matching always exists and the matching does not need to be computed explicitly.

2.5 Preprocessing

Hane et al. [16] introduced a (heuristic) preprocessing technique that can reduce the number of legs of an FAP instance and eliminate some of the ground arcs of the time space network. Here we present a small generalization of this preprocessing technique and show how it can be applied to our new MIP model and local search heuristics.

The main motivation for this preprocessing method is the fact, that airline experts prefer schedules that, totally for all subfleets in $\mathcal{F}$, use as few as possible aircraft at *small* stations. Extra aircraft should rather wait at large stations (hubs) where they can more easily be used in the case of schedule disruptions.

The minimum number of aircraft[7] needed at a station can be determined by looking at the schedule when it is operated by *one* artificial subfleet f^*. The artificial subfleet can fly every leg l "faster" than any real subfleet in $\mathcal{F}$, e.g. $[t^{dep}_{l,f^*}, t^{arr}_{l,f^*}] = \bigcap_{f\in\mathcal{F}_l}[t^{dep}_{l,f}, t^{arr}_{l,f}]$. The value of the waiting function $W_{s,f^*}(-\infty)$ at the beginning of the planning period gives you the minimal number of aircraft needed at station s and, more importantly, the islands of W_{s,f^*} define intervals in which *all* connections of a real schedule must lie if it wants to use as few aircraft at station s as the artificial subfleet f^*.

Hane et al. suggested two methods how to reduce the size of a time space network. First of all two legs k and l should be combined if they are the only legs in an island of

[6] This can only happen during the construction of a leg sequence. In that case the leg sequence is rejected.

[7] Or, to be more precise, a lower bound.

W_{s,f^*}. If you want to respect the island structure of W_{s,f^*} for your schedules k and l must always be flown by one aircraft successively and so they can be joined reducing the number of legs by one. Secondly, you can delete all ground arcs for all subfleets in the time space network that correspond to ground arcs that would run *between* islands of W_{s,f^*}. The waiting function tells us, that the flow of these ground arcs must be zero in order to achieve the minimal number of aircraft, so they can be deleted.

We propose a third reduction method that allows us to join legs that belong to islands with more than two legs. It is a generalization of the first method described above. For each island of W_{s,f^*} we compute a complete matching between arriving and departing legs. Two legs k and l are connected by an arc if there exists at least one *real* subfleet $f \in \mathcal{F}$ that can fly k and l successively[8]. Then we test every arc of the complete matching if it is part of *every* possible complete matching by removing it from the bipartite graph and trying to find an alternative complete matching. Such arcs, that are contained in every complete matching, join legs that can be combined.

The idea of joining legs can be applied to any solution method for the FAP. The idea of deleting ground arcs can also be naturally applied to our new MIP model that deals with connection dependent ground times. Besides deleting ground arcs, we can additionally remove all connection arcs that cross islands of W_{s,f^*}.

The idea of deleting ground arcs can also be looked at from a different perspective: it can be interpreted as splitting stations into ”number of islands” many independent stations. If we enforce that the number of waiting aircraft at the beginning and end of these new stations is zero for all subfleets (except for the beginning of the first and the end of the last island), we can use this idea also in our local search heuristics. By this the number of stations (and waiting functions) increases but that is negligible because the number of legs per station decreases, which speeds up the updating of the waiting functions and matchings.

The preprocessing methods described here are heuristic ones as they restrict the solution space and therefore may cause a loss of profit. But as they enforce a desired feature on an FAP solution and the loss of profit generally is quite low it rather adds to the (non-monetary) quality of the produced assignments. Furthermore these techniques can significantly reduce the run times needed for optimization, especially when using MIP based solution methods.

3 The Repair Game

3.1 Definitions

We define the Repair Game via its implicit game tree. Its examination gives us a measure for the robustness of a plan and on the other hand it presents us concrete operation recommendations.

Definition 1. *(Game Tree)*
For a rooted tree $T = (V, E)$, E being a subset of $V \times V$, let $L(T) \subset V$ be the set

[8] If no such complete matching exists, the minimal number of aircraft cannot be achieved by the real subfleets and preprocessing should not be applied to that station at all. So the matching computation also serves as a (necessary) test for validity.

of leafs of T*. In this paper, a* game tree $G = (V, E, h)$ *is a rooted tree (V,E), where* $V = V_{MAX} \dot{\cup} V_{AVG}$ *and* $h : V \to I\!N_0$.

Nodes of a game tree G represent positions of the underlying game, and edges move from one position to the next. The classes V_{MAX} and V_{AVG} represent two players MAX and AVG and for a node/position $v \in V_i$ the class V_i determines the player i who must perform the next move.

Definition 2. *(*Wmaxav *Value)*
Let $G = (V, E, h)$ *be a game tree and* $w_v : N(v) \to [0, 1]$ *be weight functions for all* $v \in V_{AVG}$*, where* $N(v)$ *is the set of all sons of a node* v*. The function* wmaxav : $V \to I\!N_0$ *is inductively defined by*

$$\text{wmaxav}(v) := \begin{cases} h(v) & \textit{if } v \in L(G) \\ \max\{\text{wmaxav}(v') \mid v' \in N(v)\} & \textit{if } v \in V_{MAX} \setminus L(G) \\ \sum_{v' \in N(v)} (w_v(v') \cdot \text{wmaxav}(v')) & \textit{if } v \in V_{AVG} \setminus L(G) \end{cases}$$

Definition 3. *(Repair Game)*
The goal of the Repair Game = *(G,p,g,f,s) is the calculation of* wmaxav(r) *for a special game tree* $G = (V, E, g + f)$ *with root* r *and uniform depth* t*;* $p \in L(G)$ *is a special leaf,* g, f *and* s *are functions. The game tree has the following properties:*

- *Let* $P = (r = v_1, v_2, \ldots, p = v_t) \in V^t$ *be the unique path from r to p. P describes a traditional, original plan.*
- V *is partitioned into sets* $S_1, \ldots, S_n, |V| \geq n \geq t$ *by the function* $s : V \to \{S_i\}_{1 \leq i \leq n}$*. All nodes which belong to the same set* S_i *are in the same* state of the system — *e.g. in aircraft scheduling: which aircraft is where at which point of time —, but they differ in the histories which have led them into this state.*
- $g : \{S_i\}_{1 \leq i \leq n} \to I\!N_0$ *defines the expected future costs for nodes depending on their state; for the special leaf p holds g(s(p)) = 0*
- $f : \bigcup_{1 \leq \tau \leq t} \{V\}^\tau \to I\!N_0$ *defines the induced repair-costs for every possible (sub)path in* (V, E) *; every sub-path* P' *of* P *has zero repair-costs, f(P') = 0*
- *the node evaluation function* $h : V \to I\!N_0$ *is defined by* $h(v) = g(s(v)) + f(r \ldots v)$*; note that* $h(p) = 0$ *holds by the definition of g and f*

The intuition behind this paragraph is as follows. We assume that an airline company has available a flight plan, and all other data which are necessary for the computation of an optimal flight plan. Especially, there are passenger desires for origin-destination pairs, costs for performing the flights etc. During the operations, however, so called disruptions occur, which express the fact that the company did not know the necessary data exactly, but that it based all its planning on estimations. Nevertheless, the original plan has got a value of its own, because with its release, agreements with customers, with partners and with others have been based on this plan. Therefore, after a disruption, the airline company is interested to find back to the plan as soon as possible. But at this moment, because many possible ways how the future may continue can be described, it does not only consider estimated data for the operations, but starts a chess-like game against Nature. How can we behave, what can Nature then do, what can we do thereafter, etc.

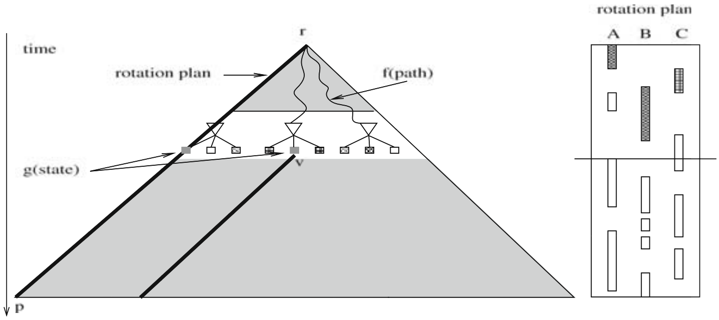


Fig. 5. The Repair Game Tree

The connection to the FAP problem is via the MIP formulation (13) to (21) in section 2.3. Let the time horizon for the problem be divided into T many intervals, and let us assign a further index t to each variable. The index $t \in T$ indicates in which time step t the leg's uncertain information is determined. If we pick up a certain time point $t' \in T$, duration and existence of the legs will be determined for all $t'' \leq t'$, but will belong to discrete probability distributions for all t'' with $t' < t'' \leq T$.

3.2 Interpretation and Airline Example

A planning team of e.g. an airline company starts the game with the construction of a traditional plan for its activities. The path P represents this planned schedule, which also is the most expected path in the time-funnel, and which interestingly gets an additional value of its own, as soon as it is generated. It is small, can be communicated, and as soon as a customer or a supplier has received the plan, each change of the plan means extra costs for the change. Disruptions in airline transportation systems can now prevent airlines from executing their schedules as planned. As soon as a specific disruption occurs, the MAX-player will select a repairing sub-plan such that the profit is maximized and the repair costs plus the expected future repair costs are minimized.

As the value of a game tree leaf v depends on how 'far' the path $(r, \ldots, v)$ is away from P, it will not be possible to identify system states (where the aircrafts are at a specific time) with tree nodes. Therefore, the tree nodes V are partitioned into $S_1 \dot{\cup} \ldots \dot{\cup} S_n$. In S_i all those nodes are collected which belong to the same state, but have different histories. All nodes in the same state S_i have the same expected future costs. These costs are estimated by the function g. The function f evaluates for an arbitrary partial path, how far it is away from the level-corresponding partial path of P. Inner nodes of the game tree are evaluated by the Wmaxav function.

Figure 5 shows a rotation plan at the right. Aircrafts A, B, and C are either on ground, or in the air, which is indicated by boxes. A shadowed box means that the original plan has been changed. The time goes from the top down. The left part of the figure shows a game tree, where the leftmost path corresponds to the original plan P. When a disruption occurs, we are forced to leave the plan, but hopefully we can return to it at some node v. The fat path from node v downward is the same as in the original plan.

Thus, at node v, we have costs for the path from the root r to v, denoted by $f(r \dots v)$ and future expected costs, denoted by $g(s(v))$. If we follow the original plan from the beginning on, we will have the same expected costs at the time point represented by v, but different path costs. The only node with zero costs typically is the special leaf node p of the original plan.

3.3 Heuristic Procedure

The aim is to compare two different repair approaches with each other. The first one is called 'Myopic MIP' solver and it repairs a plan after a disruption with the help of a slightly modified time-space network such that the solution of the associated network flow problem is an optimal one, under the assumption that no more disruptions will ever occur. This procedure represents traditional deterministic planning. The only difference is that it uses a modified cost function because the repair costs are mainly determined by the changes on the original plan, rather than by leg profits.

The second procedure, called 'T3', is one which plays the Repair Game. It compares various solutions of the modified time-space network flow problem and examines various scenarios which might occur in the near future. The forecast procedure makes use of the dynamics time-locally around time t and $t + d$ as follows: Instead of generating only one myopic MIP optimal solution for the recovery, we generate several ones. They are called our *possible moves*. A simple, certainly good heuristic is to demand that these solutions have nearly optimal costs concerning the cost function which minimizes the cost for changes. For all of these possible moves, we inspect what kind of relevant disruptions can come within the next d minutes. On all these scenarios, we repair the plan again with the help of the myopic MIP solver, which gives us value estimations for the arisen scenarios. We weight the scenarios according to their probabilities, and minimize over the expected values of the scenarios. Concerning the new part of the plan we have not optimized the costs over expected input data, but we have approximately minimized the expected costs over possible scenarios. The following algorithm is a simplified version of the algorithm shown in [3].

```
value wmaxav(node $v$, value $\alpha$, value $\beta$)
generate all successors $v_1, \dots, v_b$ of $v$; let $b$ be the number of successors
value $val := 0$;
**if** $b = 0$ return $h(v)$ /* (leaf eval)*/
**for** $i := 1$ **to** $b$
  **if** $v$ is MAX-node {
    $\alpha := \max(\alpha, \text{wmaxav}(v_i, \alpha, \beta))$; **if** $\alpha \geq \beta$ return $\beta$; **if** $i = b$ return $\beta$
  } **else** **if** $v$ is AVG-node { // let $w_1, .., w_b$ be the weights of $v_1, .., v_b$
    $\alpha' := \max(L, (\alpha - \text{val} - U \cdot \sum_{j=i+1}^{b} w_j)/w_i)$;
    $\beta' := \min(U, (\beta - \text{val} - L \cdot \sum_{j=i+1}^{b} w_j)/w_i)$;
    $val += \text{wmaxav}(v_i, \alpha', \beta') \cdot w_i$;
    **if** $val + L \cdot \sum_{j=i+1}^{b} w_j \geq \beta$ return $\beta$; **if** $val + U \cdot \sum_{j=i+1}^{b} w_j \leq \alpha$ return $\alpha$
    **if** $i = b$ return $val$
  }
```

Here, α and β are parameters, used by the recursive algorithm for cutting parts of the search tree. Typically, $\alpha := -\infty$ and $\beta := \infty$ in first main procedure call. L is a global lower bound for the objective, U a corresponding upper bound.

4 Experimental Evaluation

4.1 FAP with Sequence Dependent Ground Times

We investigated the performance (running time and solution quality) of our algorithms on several sets of real-world data. The instances are problems of recent summer and winter schedules from major airlines provided to us by our industrial partner Lufthansa Systems. Due to confidentiality only percentage results on the objective values (profit) can be shown. Our experiments include a comparison of the effects of the preprocessing techniques presented in section 2.5 and a comparison between the performance of FAPs with and without connection dependent ground times.

Table 1. Properties of the problem instances tested

FAP	legs	subfleets	stations	alternatives
A	6287	8	96	2.9
B	5243	23	76	1.7
C	5306	21	76	4.8
D	5186	20	71	4.6

Table 1 lists the sizes of the four problem instances used in this evaluation. The column ”alternatives” contains the average number of different subfleets a leg can be assigned to. Instance B is special because its alternative-value is very low. Many of the legs are fixed to one subfleet. This instance emerged form a scenario where only the legs arriving or departing at one specific station should be optimized.

All tests were executed on a PC workstation, Pentium III 933 MHz, with 512 MByte RAM running under RedHat Linux 7.3. We used two randomized heuristics using the neighborhood presented in section 2.4, a hill climbing (HC) and a simulated annealing (SA) algorithm, implemented in C. The HC and SA results show the average value of 5 runs. Our new MIP model was solved by the branch and bound IP-solver of CPLEX 7.5. The root node was computed using the interior-point barrier-method and the sub nodes were processed by the dual simplex algorithm. The IP-solver was used in a heuristic configuration terminating as soon as the first valid (integral) solution was found[9]. Therefore also for the MIP approach a solution quality is given which corresponds to the IP-gap reported by CPLEX. For the CPLEX-runs both, the root relaxation time and the total run time, are given. All run times are given in seconds.

Table 2 shows the run times and solution qualities of our algorithms on the benchmark set. HC is fastest but has the worst solution quality. MIP computes near optimal solutions in reasonable time, but normally is the slowest approach. Only on instance B it is able to outperform SA due to the very special structure of instance B mentioned

[9] The computation of an *optimal* solution can last days despite the small IP-gaps.

Table 2. Running time and solution quality of Hill Climbing, Simulated Annealing and MIP model

FAP	HC		SA		MIP		
	time	quality	time	quality	total time	root time	quality
A	16	97.99%	302	98.81%	752	54	99.90%
B	1	99.13%	13	99.84%	4	3	optimal
C	5	97.82%	84	99.40%	314	125	99.96%
D	5	98.12%	69	99.51%	229	124	99.98%

above. Finally, SA is a compromise between speed and quality. In general, it is not easy to decide, if a solution quality of say 99% is sufficiently good. On the one hand, we are dealing with huge numbers and 1% can be hundred thousands of Euro. On the other hand we are only given *estimations* of the profit that can be off by more than 10%, especially during strategic planning.

In table 3 we are presenting the effects of the preprocessing techniques of section 2.5. It contains results of four different preprocessing settings that we applied to (the representative) instance A. Preprocessing setting ”no” stands for no preprocessing, ”J” joins legs of islands that only consist of two legs, ”J*” additionally joins legs in bigger islands and ”GJ*” uses all techniques described in sections 2.5. As mentioned earlier, these preprocessing techniques are heuristic ones and therefore the first column lists the loss in profit due to the preprocessing. The following columns show the effect of preprocessing on the LP-size of our MIP model. The number of legs, number of connection, ground and flight arcs, the number of leg node equations and the total size of the LP are given. The last columns contain the run times of our MIP and SA approach. As can be

Table 3. The influence of preprocessing on our FAP algorithms

Prepro	profit loss	legs	conn. arcs	ground arcs	flight arcs	leg equ.	rows	columns	MIP time	root time	SA time
no	–	6287	8050	12054	18168	94	14544	34349	5153	179	383
J	0.39%	5102	6143	8545	14660	731	11238	26181	910	77	336
J*	0.56%	4549	6005	7163	13028	802	9693	23355	1001	59	302
GJ*	0.61%	4549	5984	6386	12953	802	9545	22611	752	54	302

Table 4. Comparison between FAPs with and without connection dependent ground times (CDGT)

	Instance A with CDGT					Instance A without CDGT				
Prepro	rows	columns	MIP time	root time	SA time	rows	columns	MIP time	root time	SA time
no	14544	34349	5153	179	383	14155	25930	3199	114	123
J	11238	26181	910	77	336	10100	18915	415	47	103
J*	9693	23355	1001	59	302	8480	16145	336	31	97
GJ*	9545	22611	752	54	302	8313	15403	360	27	95

seen, preprocessing is crucial for the MIP approach, as it is able to significantly reduce the run times. SA does not take that much of an advantage from preprocessing.

In table 4 we compare the run times of our FAP solvers on instances with and without connection dependent ground times to see the impact of adding them to the FAP. We took instance A and replaced its connection dependent ground times $g(k,l,f)$ by classical leg dependent ground times $g(k,f) = \min_{l \in \mathcal{L}} g(k,l,f)$. Thereby we built an instance without connection dependent ground times similar in size and structure to instance A. Test runs were executed using the four different preprocessing settings from above. Note that our MIP approach transforms into a regular time space network for FAPs without connection dependent ground times. The table shows that the solution times increase only by a factor of 2 to 3 by introducing connection dependent ground times, both for the MIP and SA approach.

4.2 Results Incorporating Uncertainties

Experimental Setup. In accordance with our industrial partner, we built a simulator in order to evaluate different models and algorithms The simulator is less detailed than e.g. SimAir [29], but we agreed that it is detailed enough to model the desired part of the reality. Furthermore, it is simple enough such that the occurring problems are computationally solvable.

First of all, we discretize the time of the rotation plan into steps of $d = 15$ minutes. Every departure in the current rotation plan is a possible event point, and all events inside one d minute period are interpreted as simultaneous. When an event occurs, the leg which belongs to that event can be disrupted, i.e. it can be delayed by 30, 60, or 120 minutes, or it can be canceled with certain probabilities. Let t be the present point of time. The simulator inspects the events between t and $t + d$, and informs a repair *engine* (i.e. the software which uses a repair-algorithm/heuristic) about the new disruptions, waits for a corrected rotation plan, and steps d minutes forward.

Influence of Look-Ahead. The basis for our simulations is a major airline plan plus the data which we need to build the plan and its repairs. The plan consists of 20603 legs, operated by 144 aircrafts within 6 different subfleets. The MIP for this plan has 220460 columns, 99765 rows, and 580793 non-zeros. Partitioned into single days, the resulting partial plan for any single day consists of a corresponding smaller number of columns and non-zeros.

All experiments are performed on a 4-node-cluster of the Paderborn University. Each node consists of two Pentium IV/2.8 GHz processors on a dual processor board with 1 GB of main memory. The nodes are connected by a Myrinet high speed interconnection network. The cluster runs under the RedHat Linux operating system.

We simulated 14 days, and we divided this time period into 14 pieces such that we arrived at a test set with 14 instances. Moreover, time is divided into segments of 15 minutes, and everything happening in one 15 minute block is assumed to be simultaneous. We compare the behavior of the 'myopic MIP' and the 'T3' engines. We appropriately choose the probabilities for disruptions to $(0.003/0.04/0.16/0.24)$ for (leg cancellation / delay of 120 minutes / delay of 60 minutes / delay of 30 minutes), and control the performance of the objective function $c(TIM, ECH, CNL,$

$revenue) = 50 \cdot TIM + 10000 \cdot ECH + 100000 \cdot CNL - revenue$, TIM being time-shifts, ECH meaning that the aircraft type of a leg had to be changed, CNL being the number of necessary cancellations and revenues being the traditional cost measure for the deterministic problem. A test run consists of about 1400 single decisions, and first tests showed benefits of more than three percent cost reductions over the myopic solver (cf. table 5).

Table 5. Simulation results for probabilities $0.003/0.04/0.16/0.24$

Date	MIP					T3					Δ
	TIM	CNL	ECH	profits	c	TIM	CNL	ECH	profits	c	
01/03	12210	8	2	-684	1431184	12120	8	2	-11374	1437374	-6190
01/04	11460	6	1	2028	1180972	11550	4	0	1145	976355	204617
01/05	10950	6	4	-24978	1212478	11340	8	5	-20407	1437407	-224929
01/06	13830	8	1	-5654	1507154	13470	8	1	-2567	1486067	21087
01/07	10530	7	2	-28385	1274885	10950	7	3	-33248	1310748	-35863
01/08	6000	4	4	-49501	789501	5430	2	4	-49784	561284	228217
01/09	11640	4	2	-29410	1031410	11570	4	2	-20977	1019477	6933
01/10	11730	6	3	-20403	1236903	11130	6	3	-25713	1212213	24690
01/11	12060	8	0	2003	1400997	12150	6	3	12119	1225381	175616
01/12	12030	6	2	-1971	1223471	11790	8	2	974	1408526	-185055
01/13	12630	8	0	580	1430920	12570	6	1	1036	1237464	193456
01/14	10410	6	4	-38488	1198988	10020	6	4	-36133	1177133	21855
01/15	5790	2	0	-1000	490500	5760	2	0	-1000	489000	1500
01/16	12270	5	0	-133	1113633	12090	4	0	257	1004243	109390
sums	153540	84	25	-195996	16522996	152040	79	30	-185672	15987672	535324

The columns 2 to 6 of table 5 show the results for the myopic MIP-solver, the columns 7 to 11 belong to the T3-engine. The Δ-column shows the cost differences of the two engines. Positive values indicate that the T3-engine causes less extra-costs than the myopic MIP solver. Altogether, T3 spares about 3.35% in this example. An interesting observation is that we can detect the Sundays on 8^{th} and 15^{th} January. Disruptions have significantly less consequences at these days, because the schedules on Sundays are less vulnerable.

Although a benefit of more than three percent looks already promising, statistical significance cannot be read out of a single test run. Although the engines make nearly 100 decisions per simulated day, these decisions cannot be taken as a basis for significance examinations, because the single decisions are part of a chain and not independent from each other. The results of the single days seem to form no normal distribution and, moreover, depend on the structure of the original plan. Therefore, we made further test runs and grouped those outcomes of each simulated week to one. We measure the averaged per-day profit of a week, in absolute numbers (see Table 6). The question of interest is whether or not the T3-Engine performs better then the Myopic-MIP engine, no matter how large the difference is. In table 6, the benefit of the T3-engine over the Myopic-MIP engine is presented for 12 sample points. There, an entry *week i / run j* describes the averaged per-day profit of the T3-procedure over the myopic MIP-procedure, averaged over week i, at simulation run j. Positive values are in favor of the T3-procedure.

Table 6. The daily-average profit of T3 over Myopic, two different weeks, six measure points each

	run 1	run 2	run 3	run 4	run 5	run 6
week 1	2320	27696	32261	-9238	-15799	13150
week 2	11040	48778	11580	-1253	9144	8389

The average value of these numbers is 11505 units, the standard deviation is 17897. If we assume a normal distribution, we can conclude with the help of the t-distribution that with probability 95% the expected benefit of the T3-engine over the myopic MIP-engine is greater or equal to 2309 units.

Parallelization. The basic idea of our parallelization is to use dynamic load balancing and to decompose the search tree, to search parts of the search tree in parallel and to balance the load dynamically with the help of the work stealing concept. This works as follows: First, a special processor P_0 gets the search problem and starts performing the wmaxav algorithm as if it would act sequentially. At the same time, the other processors send requests for work, the REQUEST message, to other randomly chosen processors. When a processor P_i that is already supplied with work, catches such a request, it checks, whether or not there are unexplored parts of its search tree, ready for evaluation. These unexplored parts are all rooted at the right siblings of the nodes of $P_i's$ search stack. Either, P_i sends back that it cannot serve with work with the help of the NO-WORK message, or it sends such a node (a position in the search tree etc.) to the requesting processor P_j. That is done with the help of the WORK message. Thus, P_i becomes a master itself, and P_j starts a sequential search on its own. The master/worker relationships are dynamically changed during the computation. A processor P_j being a worker of Q at a certain point of time may become the master of Q at another point of time. In general, processors are masters and workers simultaneously. If, however, a processor P_j has evaluated a node v, but a sibling of v is still under examination of another processor Q, P_j will wait until Q sends an answer message. When P_j has finished its work (possibly with the help of other processors), it sends an ANSWER message to P_i. The master-worker relationship between P_i and P_j is released, and P_j is idle again. It again starts sending requests for work into the network.

When a processor P_i finds out that it has sent a wrong local bound to one of its workers P_j, it makes a WINDOW message follow to P_j. P_j stops its search, corrects the window and starts its old search from the beginning. If the message contained a cutoff, P_j just stops its work. A processor P_i can shorten its search stack due to an external message, when e.g. a CUTOFF message comes in which belongs to a node near the root. In absence of deep searches and advanced cutting techniques, however, CUTOFF and WINDOW messages did not occur in our experiments.

In many applications, the computation at each node is fast in relation to the exchange of a message. E.g. there are chess programs which do not use more than 1000 processor cycles per search node, including move generation, moving end evaluating the node. In the application presented here, the situation is different. The sequential computations at each node takes several seconds if not even minutes, such that the latency of messages is not remarkably important. Therefore, the presented application is certainly well

Table 7. Speedups

# proc	simtime day 1	SPE day1	simtime day 2	SPE day2	simtime day 3	SPE day3
1	226057	1	193933	1	219039	1
2	128608	1.76	111612	1.73	126915	1.73
4	68229	3.31	59987	3.23	66281	3.30
8	46675	4.84	40564	4.78	46065	4.75

suited for grid computing, as well. However, it is necessary to overlap communication and computations. We decoupled the MIP solver from the rest of the application and assigned a thread to it.

In distributed systems, messages can be delayed by the system. Messages from the past might arrive, which are outdated. Therefore, for every node v a processor generates a local unique ID, which is added to every message belonging to node v. Thus, we are always able to identify misleading results and to discard invalid messages. We refer to [11] for further details.

Speedups. We measure speedups of our program with the help the first three days of the test set which consists of the 14 single days, mentioned above. The time for simulation of the days using one processor is compared with the running time of several processors. The speedup (SPE) is the sum of the times of the sequential version divided by the sum of the times of a parallel version [12]. Each test day consists of 96 single measure points. Table 7 shows the speedups which we could achieve. We are quite satisfied with these speedups because they bring the necessary computations on a real-time level. Of course, the question arises how the work load and the search overhead behave. As the sequential and the parallel version do indeed exactly the same, search overhead does not exist. Neither the costs for communication are relevant. The only reason that we cannot achieve the full speedup are the large sequential computing periods, caused by the commercial MIP solver.

5 Conclusions

The first major contribution of this paper shows how an important operational restriction, connection dependent ground times, can be incorporated into the fleet assignment problem and that this extended FAP can be solved for real-world problem instances. We presented and evaluated three different optimization methods, two local search based (HC and SA) and one MIP based approach, that have different characteristics concerning run time and solution quality. HC is fast but produces low quality solutions, MIP calculates near optimal solutions but needs longer and SA is a compromise between speed and quality.

Now, with the help of this framework, it should be possible to include the through-assignment problem, which is part of the aircraft rotation building optimization step. Furthermore certain punctuality restrictions, like forbidding successive connections without time buffers, can be integrated.

Playing the Repair Game leads to more robust (sub-)plans in airline scheduling than traditional deterministic planning can provide. This is the second major contribution.

Our forecast strategy outperforms the myopic MIP solver by means of simulations. We have parallelized the search in order to drop the computation times to real time.

References

1. Abara, J.: Applying integer linear programming to the fleet assignment problem. Interfaces 19(4), 20–28 (1989)
2. Althöfer, I.: Root evaluation errors: How they arise and propagate. ICCA Journal 11(3), 55–63 (1988)
3. Ballard, B.W.: The *-minimax search procedure for trees containing chance nodes. Artificial Intelligence 21(3), 327–350 (1983)
4. Barnhart, C., Boland, N.L., Clarke, L.W., Shenoi, R.G.: Flight strings models for aircraft fleeting and routing. Technical report. MIT Cambridge (1997)
5. Belanger, N., Desaulniers, G., Soumis, F., Desrosiers, J., Lavigne, J.: Airline fleet assignment with homogeneity. Technical report, GERAD, Montréal (2002)
6. Berge, M.A., Hopperstad, C.A.: Demand driven dispatch: A method for dynamic aircraft capacity assignment, models and algorithms. Operations Research 41(1), 153–168 (1993)
7. Clarke, L.W., Hane, C.A., Johnson, E.L., Nemhauser, G.L.: Maintenance and crew considerations in fleet assignment. Technical report (1994)
8. Desaulniers, G., Desrosiers, J., Dumas, Y., Solomon, M.M., Soumis, F.: Daily aircraft routing and scheduling. Management Science 43(6), 841–855 (1997)
9. Ehrhoff, J., Grothklags, S., Lorenz, U.: Das Reparaturspiel als Formalisierung von Planung unter Zufallseinflüssen, angewendet in der Flugplanung. In: Proceedings of GOR conference: Entscheidungsunterstützende Systeme in Supply Chain Managment und Logistik, pp. 335–356. Physika-Verlag (2005)
10. Engell, S., Märkert, A., Sand, G., Schultz, R.: Production planning in a multiproduct batch plant under uncertainty. Preprint 495-2001, FB Mathematik, Gerhard-Mercator-Universität Duisburg (2001)
11. Feldmann, R., Mysliwietz, M., Monien, B.: Studying overheads in massively parallel min/max-tree evaluation. In: 6th ACM Annual symposium on parallel algorithms and architectures (SPAA 1994), pp. 94–104. ACM, New York (1994)
12. Flemming, P.J., Wallace, J.J.: How not to lie with statistics: the correct way to summerize benchmark results. CACM 29(3), 218–221 (1986)
13. Gertsbach, I., Gurevich, Y.: Constructing an optimal fleet for a transportation schedule. Transportation Science 11(1), 20–36 (1977)
14. Götz, S., Grothklags, S., Kliewer, G., Tschöke, S.: Solving the weekly fleet assignment problem for large airlines. In: MIC 1999, pp. 241–246 (1999)
15. Gu, Z., Johnson, E.L., Nemhauser, G.L., Wang, Y.: Some properties of the fleet assignment problem. Operations Research Letters 15, 59–71 (1994)
16. Hane, C.A., Barnhart, C., Johnson, E.L., Marsten, R.E., Nemhauser, G.L., Sigismondi, G.: The fleet assignment problem: solving a large-scale integer program. Mathematical Programming 70, 211–232 (1995)
17. Kaindl, H., Scheucher, A.: The reason for the benefits of minmax search. In: Proc. of the 11^{th} IJCAI, Detroit, MI, pp. 322–327 (1989)
18. Kniker, T.S., Barnhart, C.: Shortcomings of the conventional fleet assignment model. Technical report. MIT, Cambridge (1998)
19. Knuth, D.E., Moore, R.W.: An analysis of alpha-beta pruning. Artificial Intelligence 6(4), 293–326 (1975)

20. Koenig, S., Furcy, D., Bauer, C.: Heuristic search-based replanning. In: Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling, pp. 294–301 (2002)
21. Kouvelis, P., Daniels, R.L., Vairaktarakis, G.: Robust scheduling of a two-machine flow shop with uncertain processing times. IIE Transactions 32(5), 421–432 (2000)
22. Leon, V.J., Wu, S.D., Storer, R.h.: A game-theoretic control approach for job shops in the presence of disruptions. International Journal of Production Research 32(6), 1451–1476 (1994)
23. Nau, D.S.: Pathology on game trees revisited, and an alternative to minimaxing. Artificial Intelligence 21(1-2), 221–244 (1983)
24. Osman, I.H.: Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research 41, 421–451 (1993)
25. Papadimitriou, C.H.: Games against nature. Journal of Computer and System Science 31, 288–301 (1985)
26. Radicke, U.-D.: Algorithmen für das Fleet Assignment von Flugplänen. Verlag Shaker (1994)
27. Reinefeld, A.: An Improvement of the Scout Tree Search Algorithm. ICCA Journal 6(4), 4–14 (1983)
28. Römisch, W., Schultz, R.: Multistage stochastic integer programming: an introduction. In: Online Optimization of Large Scale Systems, pp. 581–600 (2001)
29. Rosenberger, J.M., Schaefer, A.J., Goldsman, D., Johnson, E.L., Kleywegt, A.J., Nemhauser, G.L.: Simair: A stochastic model of airline operations. In: Winter Simulation Conference Proceedings (2000)
30. Rushmeier, R.A., Kontogiorgis, S.A.: Advances in the optimization of airline fleet assignment. Transportation Science 31(2), 159–169 (1997)
31. Russel, S., Norvig, P.: Artificial Intelligence, A Modern Approach. Prentice Hall Series in Artificial Intelligence (2003)
32. Shannon, C.E.: Programming a computer for playing chess. Philosophical Magazine 41, 256–275 (1950)
33. Sharma, D., Ahuja, R.K., Orlin, J.B.: Neighborhood search algorithms for the combined through-fleet assignment model. Talk at ISMP 2000 (2000)
34. Sosnowska, D.: Optimization of a simplified fleet assignment problem with metaheuristics: Simulated annealing and GRASP. In: Pardalos, P.M. (ed.) Approximation and Complexity in Numerical Optimization. Kluwer Academic Publisher, Dordrecht (2000)
35. Subramanian, R., Sheff, R.P., Quillinan, J.D., Wiper, D.S., Marsten, R.E.: Coldstart: Fleet assignment at Delta Air Lines. Interfaces 24(1), 104–120 (1994)
36. Talluri, K.T.: Swapping applications in a daily airline fleet assignment. Transportation Science 30(3), 237–248 (1996)

Traffic Networks and Flows over Time*

Ekkehard Köhler[1], Rolf H. Möhring[2], and Martin Skutella[2]

[1] Brandenburgische Technische Universität Cottbus,
Fakultät I — Mathematik, Naturwissenschaften und Informatik,
Mathematisches Institut, PSF 10 13 44, 03013 Cottbus, Germany
ekoehler@math.tu-cottbus.de
http://www.math.tu-cottbus.de/INSTITUT/lsgdi

[2] Technische Universität Berlin,
Fakultät II — Mathematik und Naturwissenschaften,
Institut für Mathematik, Sekr. MA 5–1,
Straße des 17. Juni 136, D–10623 Berlin, Germany
{moehring,skutella}@math.tu-berlin.de
http://www.math.tu-berlin.de/~{moehring,skutella}

1 Introduction

In view of the steadily growing car traffic and the limited capacity of our street networks, we are facing a situation where methods for better traffic management are becoming more and more important. Studies [92] show that an individual "blind" choice of routes leads to travel times that are between 6% and 19% longer than necessary. On the other hand, telematics and sensory devices are providing or will shortly provide detailed information about the actual traffic flows, thus making available the necessary data to employ better means of traffic management.

Traffic management and route guidance are optimization problems by nature. We want to utilize the available street network in such a way that the total network "load" is minimized or the "throughput" is maximized. This article deals with the mathematical aspects of these optimization problems from the viewpoint of network flow theory. This is, in our opinion, currently the most promising method to get structural insights into the behavior of traffic flows in large, real-life networks. It also provides a bridge between traffic simulation [12,77] or models based on fluid dynamics [80] and variational inequalities [23], which are two other common approaches to study traffic problems. While simulation is a powerful tool to evaluate traffic scenarios, it misses the optimization potential. On the other hand, fluid models and other models based on differential equations capture very well the dynamical behavior of traffic as a continuous quantity, but can currently not handle large networks.

Traffic flows have two important features that make them difficult to study mathematically. One is 'congestion', and the other is 'time'. 'Congestion' captures the fact that travel times increase with the amount of flow on the streets, while 'time' refers to the movement of cars along a path as a "flow over time".

* A preliminary version of this paper has appeared as [59]. Some parts have been taken from the following papers: [9,29,41,51,58,61].

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 166–196, 2009.

We will therefore study several mathematical models of traffic flows that become increasingly more difficult as they capture more and more of these two features.

Section 2 deals with flows without congestion. The first part gives an introduction to the classical static network flow theory and discusses basic results in this area. The presented insights are also at the core of more complex models and algorithms discussed in later sections. The main part of the section then introduces the temporal dimension and studies flows over time without congestion. Although of indisputable practical relevance, flows over time have never attracted as much attention among researchers as their classical static counterparts. We discuss some more recent results in this area which raise hope and can be seen as first steps towards the development of methods that can cope with flows over time as they occur in large real-life traffic networks. We conclude this section with a short discussion of flows over time as they occur in evacuation planning.

In Section 3 we add congestion to the static flow model. This is already a realistic traffic model for rush-hour traffic where the effect of flow changing over time is secondary compared to the immense impact of delays due to congestion. We consider algorithms for centralized route guidance and discuss fairness aspects for the individual user resulting from optimal route guidance policies. Finally, we review fast algorithms for shortest path problems which are at the core of more complex algorithms for traffic networks and whose running time is therefore of utmost importance.

Section 4 then combines flows over time with congestion. Compared to the models discussed in earlier sections, this setting is considerably closer to real-world traffic flows. Thus, it hardly surprises that it also leads to several new mathematical difficulties. One main reason is that the technique of time expansion which still worked for the problems of Section 2 is no longer fully available. Nevertheless, it is possible to derive approximation algorithms for reasonable models of congestion, but the full complexity of this problem is by far not understood yet.

Despite the partial results and insights discussed in this article, the development of suitable mathematical tools and algorithms which can deal with large real-world traffic networks remains an open problem. The inherent complexity of many traffic flow problems constitutes a major challenge for future research.

2 Static Flows and Flows over Time

An exhaustive mathematical treatment of network flow theory started around the middle of the last century with the ground-breaking work of Ford and Fulkerson [35]. Historically, the study of network flows mainly originates from problems related to transportation of materials or goods; see, e. g., [46,54,63]. For a detailed survey of the history of network flow problems we refer to the recent work of Schrijver [87].

2.1 Static Flow Problems

Usually, flows are defined on networks (directed graphs) $G = (V, E)$ with *capacities* $u_e \geq 0$ and, in some settings, also *costs* c_e on the arcs $e \in E$. The set of nodes V is partitioned into *source* nodes, *intermediate* nodes, and *sink* nodes. On an intuitive level, flow emerges from the source nodes, travels through the arcs of the network via intermediate nodes to the sinks, where it is finally absorbed. More precisely, a *static flow* x assigns a value $0 \le x_e \le u_e$ to every arc $e \in E$ of the network such that for every node $v \in V$

$$\sum_{e\in\delta^-(v)} x_e - \sum_{e\in\delta^+(v)} x_e \begin{cases} \le 0 & \text{if } v \text{ is a source,} \\ = 0 & \text{if } v \text{ is an intermediate node,} \\ \ge 0 & \text{if } v \text{ is a sink.} \end{cases} \tag{1}$$

Here, $\delta^+(v)$ and $\delta^-(v)$ denote the set of arcs e leaving node v and entering node v, respectively. Thus, the left hand side of (1) is the net amount of flow entering node v. For intermediate nodes this quantity must obviously be zero; this requirement is usually refered to as *flow conservation constraint.* A flow x is said to *satisfy the demands and supplies* d_v, $v \in V$, if the left hand side of (1) is equal to d_v for every node $v \in V$. In this setting, nodes v with negative demand d_v (i. e., positive supply $-d_v$) are sources and nodes with positive demand are sinks. A necessary condition for such a flow to exist is $\sum_{v\in V} d_v = 0$. Observe that the sum of the left hand side of (1) over all $v \in V$ is always equal to 0.

A flow problem in a network $G = (V, E)$ with several sources $S \subset V$ and several sinks $T \subset V$ can easily be reduced to a problem with a single source and a single sink: Introduce a new *super-source* s and a new *super-sink* t. Then add an arc (s, v) of capacity $u_{(s,v)} := -d_v$ for every source $v \in S$ and an arc (w, t) of capacity $u_{(w,t)} := d_w$ for every sink $w \in T$. In the resulting network with node set $V \cup \{s, t\}$ all nodes in V are intermediate nodes and we set $d_s := \sum_{v\in S} d_v$ and $d_t := \sum_{w\in T} d_w$.

For the case of one single source s and one single sink t, the flow x is called an *s-t-flow* and the left hand side of (1) for $v = t$ is the *s-t-flow value* which we denote by $|x|$. Due to flow conservation constraints, the absolute value of the left hand side of (1) for $v = s$ also equals the flow value. In other words, all flow leaving the source must finally arrive at the sink. Ford and Fulkerson [33,35] and independently Elias, Feinstein, and Shannon [24] show in their so-called 'Max-Flow-Min-Cut Theorem' that the maximum s-t-flow value is equal to the minimum capacity of an s-t-cut. An *s-t-cut* $\delta^+(S)$ is given by a subset of vertices $S \subseteq V \setminus \{t\}$, $s \in S$, and defined as the set of arcs going from S to its complement $V \setminus S$, i. e., $\delta^+(S) := (\bigcup_{v\in S} \delta^+(v)) \setminus (\bigcup_{v\in S} \delta^-(v))$. The *capacity* of an s-t-cut is the sum of the capacities of all arcs in the cut. We give an example of a maximum s-t-flow and a matching minimum cut in Figure 1. Ford and Fulkerson [35] also observe that the Max-Flow-Min-Cut Theorem can be interpreted as a special case of linear programming duality.

Today, a variety of efficient (i. e., polynomial time) algorithms are known for computing an s-t-flow of maximum value and a corresponding minimum capacity cut. We refer to the standard textbook by Ahuja, Magnanti, and Orlin [1] for a detailed account of results and algorithms in this area. However, we mention one further structural result for s-t-flows by Ford and Fulkerson [35]. Any given

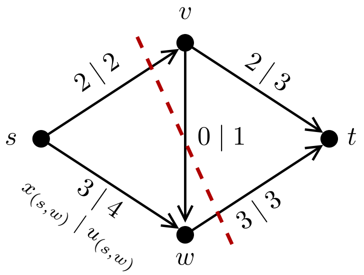


Fig. 1. A maximum s-t-flow together with a matching minimum cut in a network. The first number at each arc denotes its flow value and the second number its capacity. The depicted s-t-flow has value 5 which equals the capacity of the depicted s-t-cut $\delta^+(\{s,w\}) = \{(s,v),\, (w,t)\}$.

s-t-flow $x = (x_e)_{e \in E}$ can be decomposed into the sum of flows of value x_P on certain s-t-paths $P \in \mathcal{P}$ and flows of value x_C on certain cycles $C \in \mathcal{C}$, that is,

$$x_e \;=\; \sum_{\substack{P \in \mathcal{P} \\ e \in P}} x_P \;+\; \sum_{\substack{C \in \mathcal{C} \\ e \in C}} x_C \qquad \text{for all } e \in E.$$

Moreover, the number of paths and cycles in this decomposition can be bounded by the number of arcs, i. e., $|\mathcal{P}| + |\mathcal{C}| \le |E|$.

In the setting with costs on the arcs, the cost of a static flow x is defined as $c(x) := \sum_{e \in E} c_e\, x_e$. For given demands and supplies d_v, $v \in V$, the *minimum cost flow problem* asks for a flow x with minimum cost $c(x)$ satisfying these demands and supplies. As for the maximum flow problem discussed above, various efficient algorithms are known for this problem and a variety of structural characterizations of optimal solutions has been derived. Again, we refer to [1] for details.

A static flow problem which turns out to be considerably harder than the maximum flow and the minimum cost flow problem is the *multicommodity flow problem*. Every *commodity* $i \in K$ is given by a source-sink pair $s_i, t_i \in V$ and a prescribed flow value d_i. The task is to find an s_i-t_i-flow x_i of value d_i for every commodity $i \in K$ such that the sum of these flows obeys the arc capacities, i. e., $\sum_{i \in K} (x_i)_e \le u_e$ for all $e \in E$. So far, no combinatorial algorithm has been developed which solves this problem efficiently. On the other hand, there exist polynomial time algorithms which, however, rely on a linear programming formulation of the problem and thus on general linear programming techniques (such as the ellipsoid method or interior point algorithms); see e. g. [1].

2.2 Flows over Time

Flow variation over time is an important feature in network flow problems arising in various applications such as road or air traffic control, production systems, communication networks (e. g., the Internet), and financial flows. This characteristic is obviously not captured by the static flow models discussed in the previous section. Moreover, apart from the effect that flow values on arcs may change over time, there is a second temporal dimension in these applications: Usually, flow does not travel instantaneously through a network but requires a certain amount of time to travel through each arc. Thus, not only the amount of flow to be transmitted but also the time needed for the transmission plays an essential role.

Definition of flows over time. Ford and Fulkerson [34,35] introduce the notion of 'dynamic flows' or 'flows over time' which comprises both temporal features. They consider networks G with capacities u_e and transit times $\tau_e \in \mathbb{Z}_+$ on the arcs $e \in E$. A *flow over time* f on G with *time horizon* T is given by a collection of functions $f_e : [0, T) \to [0, u_e]$ where $f_e(\theta)$ determines the rate of flow (per time unit) entering arc e at time θ. Here, capacity u_e is interpreted as an upper bound on the rate of flow entering arc e, i. e., a capacity per unit time. Transit times are fixed throughout, so that flow on arc e progresses at a uniform rate. In particular, the flow $f_e(\theta)$ entering the tail of arc e at time θ arrives at the head of e at time $\theta + \tau_e$. In order to get an intuitive understanding of flows over time, one can associate the arcs of the network with pipes in a pipeline system for transporting some kind of fluid. The length of each pipeline determines the transit time of the corresponding arc while the width determines its capacity.

Continuous vs. discrete time. In fact, Ford and Fulkerson introduce a slightly different model of flows over time where time is discretized into intervals of unit size $[0, 1)$, $[1, 2), \ldots, [T - 1, T)$, and the function f_e maps every such interval to the amount of flow which is sent within this interval into arc e. We call such a flow over time *discrete*. Discrete flows over time can for instance be illustrated by associating each arc with a conveyor belt which can grasp one packet per time unit. The maximal admissible packet size determines the capacity of the corresponding arc and the speed and length of the conveyor belt define its transit time.

Fleischer and Tardos [31] point out a strong connection between the discrete time model and the continuous time model described above. They show that most results and algorithms which have been developed for the discrete time model can be carried over to the continuous time model and vice versa. Loosely speaking, the two models can be considered to be equivalent.

Flow conservation constraints. When considering flows over time, the flow conservation constraints (1) must be integrated over time to prohibit deficit at any node v which is not a source. Hence, for all $\xi \in [0, T)$

$$\sum_{e \in \delta^-(v)} \int_{\tau_e}^{\xi} f_e(\theta - \tau_e)\ d\theta \ - \sum_{e \in \delta^+(v)} \int_0^{\xi} f_e(\theta)\ d\theta \ \geq\ 0\ . \tag{2}$$

The first term on the left hand side of (2) is the total inflow into node v until time ξ and the second term is the total outflow out of v until time ξ. Notice that since the left hand side of (2) might be positive, temporary storage of flow at intermediate nodes is allowed. This corresponds to holding inventory at a node before sending it onward. However, we require equality in (2) for $\xi = T$ and any intermediate node v in order to enforce flow conservation in the end. If storage of flow at intermediate nodes is undesired, it can be prohibited by requiring equality in (2) for all $\xi \in [0, T)$.

As in the case of static flows, a flow over time f satisfies the demands and supplies d_v, $v \in V$, if the left hand side of (2) for $\xi = T$ is equal to d_v for every node $v \in V$. If there is only one single sink t, the left hand side of (2) for $\xi = T$ and $v = t$ is the flow value of f.

Time-expanded networks. Ford and Fulkerson [34,35] observe that flow-over-time problems in a given network with transit times on the arcs can be transformed into equivalent static flow problems in the corresponding *time-expanded network*. The time-expanded network contains one copy of the node set of the underlying 'static' network for each discrete time step θ (building a *time layer*). Moreover, for each arc e with transit time τ_e in the given network, there is a copy between each pair of time layers of distance τ_e in the time-expanded network. An illustration of this construction is given in Figure 2. Thus, a discrete flow over time in the given network can be interpreted as a static flow in the corresponding time-expanded network. Since this interrelation works in both directions, the concept of time-expanded networks allows to solve a variety of time-dependent flow problems by applying algorithmic techniques developed for static network flows; see e. g. [31]. Notice, however, that one has to pay for this simplification of the considered flow problem in terms of an enormous increase in the size of the network. In particular, the size of the time-expanded network is only pseudo-polynomial in the input size and thus does not directly lead to efficient algorithms for computing flows over time. Nevertheless, time-expanded networks are often used to solve problems in practice.

Maximum s-t-flows over time. Ford and Fulkerson [34,35] give an efficient algorithm for the problem of sending the maximum possible amount of flow from

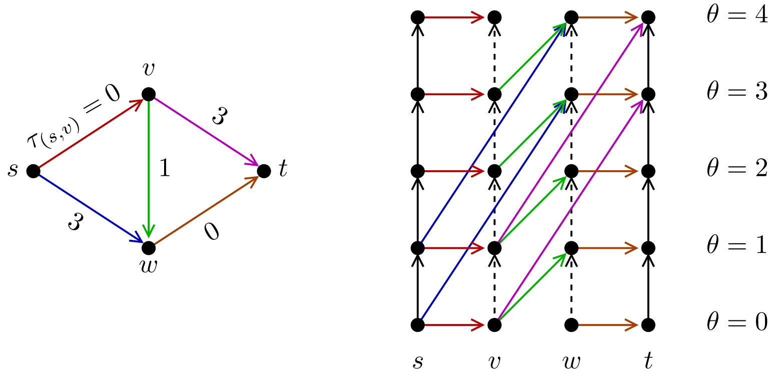


Fig. 2. A network with transit times on the arcs (left hand side) and the corresponding time-expanded network (right hand side). The vertical black arcs model the possibility to store flow at nodes.

one source to one sink within a given time horizon T. They show that this problem can be solved by one minimum-cost flow computation in the given network, where transit times of arcs are interpreted as cost coefficients. Their algorithm is based on the concept of *temporally repeated flows*. Let x be a feasible static s-t-flow in G with path decomposition $(x_P)_{P \in \mathcal{P}}$ where $\mathcal{P}$ is a set of s-t-paths. If the transit time $\tau_P := \sum_{e \in P} \tau_e$ of every path $P \in \mathcal{P}$ is bounded from above by T, the static s-t-flow x can be turned into a *temporally repeated* s-t-flow f as follows. Starting at time zero, f sends flow at constant rate x_P into path $P \in \mathcal{P}$ until time $T - \tau_P$, thus ensuring that the last unit of flow arrives at the sink before time T. An illustration of temporally repeated flows is given in Figure 3.

Feasibility of f with respect to capacity constraints immediately follows from the feasibility of the underlying static flow x. Notice that the flow value $|f|$ of f, i. e., the amount of flow sent from s to t is given by

$$|f| \;=\; \sum_{P \in \mathcal{P}} (T - \tau_P)\, x_P \;=\; T\,|x| - \sum_{e \in E} \tau_e\, x_e \quad . \tag{3}$$

The algorithm of Ford and Fulkerson computes a static s-t-flow x maximizing the right hand side of (3), then determines a path decomposition of x, and finally returns the corresponding temporally repeated flow f. Ford and Fulkerson show that this temporally repeated flow is in fact a maximum s-t-flow over time by presenting a matching minimum cut in the corresponding time-expanded network. This cut in the time-expanded network can be interpreted as a *cut over time* in the given network, that is, a cut wandering through the network over time from the source to the sink.

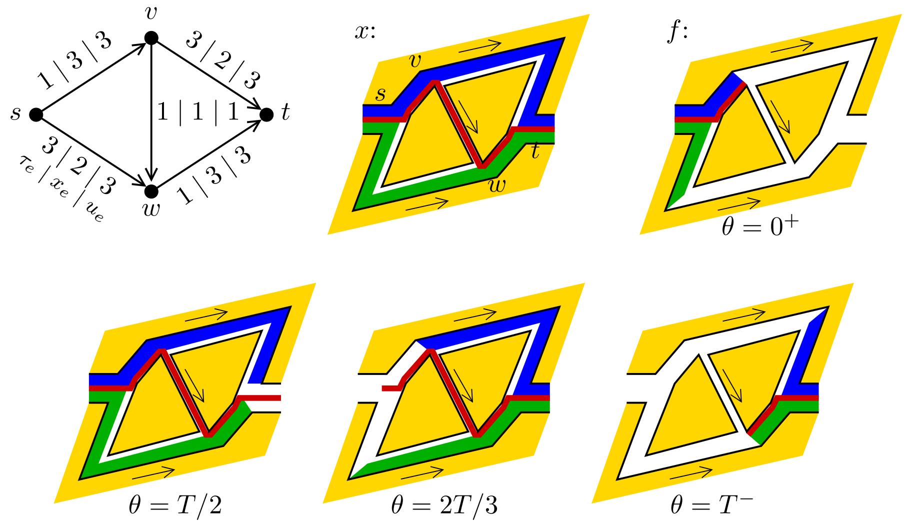


Fig. 3. A static s-t-flow x and 4 snapshots of the corresponding temporally repeated s-t-flow f

A problem closely related to the one considered by Ford and Fulkerson is the *quickest s-t-flow problem*. Here, instead of fixing the time horizon T and asking for a flow over time of maximal value, the value of the flow (demand) is fixed and T is to be minimized. This problem can be solved in polynomial time by incorporating the algorithm of Ford and Fulkerson into a binary search framework. Burkard, Dlaska, and Klinz [15] observed that the quickest flow problem can even be solved in strongly polynomial time.

More complex flow over time problems. As mentioned above, a static flow problem with multiple sources and sinks can easily be reduced to one with a single source and a single sink. Notice, however, that this approach no longer works for flows over time. It is in general not possible to control the amount of flow which is sent on the arcs connecting the super-source and super-sink to the original nodes of the network. Hoppe and Tardos [49] study the *quickest transshipment problem* which asks for a flow over time satisfying given demands and supplies within minimum time. Surprisingly, this problem turns out to be much harder than the special case with a single source and sink. Hoppe and Tardos give a polynomial time algorithm for the problem, but this algorithm relies on submodular function minimization and is thus much less efficient than for example the algorithm of Ford and Fulkerson for maximum s-t-flows over time.

Even more surprising, Klinz and Woeginger [55] show that the problem of computing a minimum cost s-t-flow over time with prespecified value and time horizon is NP-hard. This means that under the widely believed conjecture P≠NP, there does not exist an efficient algorithm for solving this problem. The cost of a flow over time f is defined as

$$c(f) \ := \ \sum_{e \in E} c_e \int_0^T f_e(\theta) \ d\theta \ .$$

In fact, Klinz and Woeginger prove that the problem is already NP-hard on very special series-parallel networks. The same hardness result holds for the *quickest multicommodity flow problem*; see Hall, Hippler, and Skutella [41]. On the other hand, Hall et al. develop polynomial time algorithms for solving the problem on tree networks. An overview of the complexity landscape of flows over time is given in Table 1.

Approximation results. Fleischer and Skutella [27] generalize the temporally repeated flow approach of Ford and Fulkerson in order to get an approximation algorithm for the quickest multicommodity flow problem with bounded cost. A key insight in their result is that by taking the temporal average of an optimal multicommodity flow over time with time horizon T, one gets a static flow in the given network which is T-length-bounded, i. e., it can be decomposed into flows on paths whose transit time is bounded by T. While it is initially not clear how to compute a good multicommodity flow over time efficiently, a length-bounded static flow mimicking the static 'average flow' can be obtained in polynomial time (if the length bound is relaxed by a factor of $1 + \epsilon$). The computed static

Table 1. The complexity landscape of flows over time in comparison to the corresponding static flow problems. The third column 'transshipment' refers to single-commodity flows with several source and sink nodes. The quoted approximation results hold for the corresponding quickest flow problems.

	s-t-flow	transshipment	min-cost flow	multicommodity flow
(static) flow	poly	poly (≃ s-t-flow)	poly	poly (≃ LP)
flow over time with storage	poly [34] (≃ min-cost flow)	poly [49] (≃ subm. func.)	pseudo-poly NP-hard [55] FPTAS [28,29]	pseudo-poly NP-hard [41] FPTAS [28,29]
flow over time without storage				strongly NP-hard [41] 2-approx. [27,29]

multicommodity flow can then be turned into a temporally repeated flow in a similar way as in the algorithm of Ford and Fulkerson. Moreover, the time horizon of this temporally repeated flow is at most $(2+\epsilon)T$ and therefore within a factor of $2+\epsilon$ of the optimum. Martens and Skutella [70] make use of this approach to approximate s-t-flows over time with a bound on the number of flow-carrying paths (*k-splittable flows over time*).

Fleischer and Skutella [27,28] also show that better performance ratios can be obtained by using a variant of time-expanded networks. In *condensed* time-expanded networks, a rougher discretization of time is used in order to get networks of polynomial size. However, in order to discretize time into steps of larger length $L > 1$, transit times of arcs have to be rounded up to multiples of L. This causes severe problems in the analysis of the quality of solutions computed in condensed time-expanded networks. Nevertheless, one can show that the solution space described by a condensed time-expanded network is only slightly degraded. This yields a general framework to obtain fully polynomial time approximation schemes for various flow-over-time problems like, for example, the quickest multicommodity flow problem with costs. Moreover, this approach also yields more efficient algorithms for solving flow over time problems in practical applications.

For the case that arc costs are proportional to transit times, Fleischer and Skutella [28] describe a very simple fully polyomial-time approximation scheme based on capacity scaling for the minimum cost s-t-flow over time problem.

For a broad overview of flows over time we refer to the survey papers by Aronson [3], Powell, Jaillet, and Odoni [79], Kotnyek [64], and Lovetskii and Melamed [68]. We also refer to the PhD thesis of Hoppe [47] for an easily accessible and detailed treatment of the topic based on the *discrete time model.* Skutella [90] presents a treatment of flows over time for the purpose of teaching in an advanced course.

2.3 Earliest Arrival Flows

In typical evacuation situations, the most important task is to get people out of an endangered building or area as fast as possible. Since it is usually not known how long a building can withstand a fire before it collapses or how long a

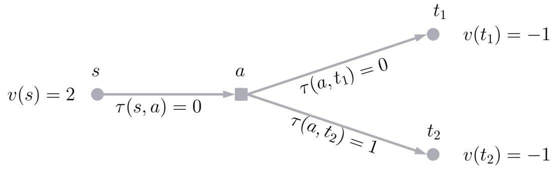


Fig. 4. A network with one source and two sinks with unit demands for which an earliest arrival flow does not exist. All arcs have unit capacity and the transit times are given in the drawing. Notice that one unit of flow can reach sink t_1 by time 1; in this case, the second unit of flow reaches sink t_2 only by time 3. Alternatively we can send one unit of flow into sinks t_1 and t_2 simultaneously by time 2. It is impossible to fulfill the requirement, that both flow units must have reached their sink by time 2 and one of them must have already arrived at time 1.

dam can resist a flood before it breaks, it is advisable to organize an evacuation such that as much as possible is saved no matter when the inferno will actually happen. In the setting of flows over time, the latter requirement is captured by so-called earliest arrival flows.

Shortly after Ford and Fulkerson introduced flows over time, the more elaborate *earliest arrival s-t-flow problem* was studied by Gale [37]. Here the goal is to find a single s-t-flow over time that simultaneously maximizes the amount of flow reaching the sink t up to any time $\theta \geq 0$. A flow over time fulfilling this requirement is said to have the *earliest arrival property* and is called *earliest arrival s-t-flow*. Gale [37] showed that earliest arrival s-t-flows always exist. Minieka [75] and Wilkinson [94] both gave pseudopolynomial-time algorithms for computing earliest arrival s-t-flows based on the Successive Shortest Path Algorithm [53,50,16]. Hoppe and Tardos [48] present a fully polynomial-time approximation scheme for the earliest arrival s-t-flow problem that is based on a clever scaling trick.

In a network with several sources and sinks with given supplies and demands, flows over time having the earliest arrival property do not necessarily exist [30]. Baumann and Skutella [9] give a simple counterexample with one source and two sinks; see Figure 4.

For the case of several sources with given supplies and a single sink, however, earliest arrival transshipments do always exist. This follows, for example, from the existence of lexicographically maximal flows in time-expanded networks; see, e.g., [75]. We refer to this problem as the *earliest arrival transshipment problem*. Hajek and Ogier [40] give the first polynomial time algorithm for the earliest arrival transshipment problem with zero transit times. Fleischer [30] gives an algorithm with improved running time. Fleischer and Skutella [29] use condensed time-expanded networks to approximate the earliest arrival transshipment problem for the case of arbitrary transit times. They give an FPTAS that approximates the time delay as follows: For every time $\theta \geq 0$ the amount of flow that should have reached the sink in an earliest arrival transshipment by time θ, reaches the sink at latest at time $(1+\varepsilon)\theta$. Tjandra [91] shows how to compute earliest arrival transshipments in networks with time dependent supplies and capacities in time polynomial in the time horizon and the total supply at sources. The resulting running time is thus only pseudopolynomial in the input size.

Earliest arrival flows and transshipments are motivated by applications related to evacuation. In the context of emergency evacuation from buildings, Berlin [13] and Chalmet et al. [20] study the quickest transshipment problem in networks with multiple sources and a single sink. Jarvis and Ratliff [52] show that three different objectives of this optimization problem can be achieved simultaneously: (1) Minimizing the total time needed to send the supplies of all sources to the sink, (2) fulfilling the earliest arrival property, and (3) minimizing the average time for all flow needed to reach the sink. Hamacher and Tufecki [45] study an evacuation problem and propose solutions which further prevent unnecessary movement within a building.

Baumann and Skutella [9] (see also [7]) present a polynomial time algorithm for computing earliest arrival transshipments in the general multiple-source single-sink setting. All previous algorithms rely on time expansion of the network into exponentially many time layers. Baumann and Skutella first recursively construct the earliest arrival pattern, that is, the piece-wise linear function that describes the time-dependent maximum flow value obeying supplies and demands. The algorithm employs submodular function minimization within the parametric search framework of Megiddo [71,72]. As a by-product, a new proof for the existence of earliest arrival transshipments is obtained that does not rely on time expansion. It can finally be shown that the earliest arrival pattern can be turned into an earliest arrival transshipment based on the quickest transshipment algorithm of Hoppe and Tardos [49].

The running time of the obtained algorithm is polynomial in the input size plus the number of breakpoints of the earliest arrival pattern. Since the earliest arrival pattern is more or less explicitly part of the output of the earliest arrival transshipment problem, the running time of the algorithm is polynomially bounded in the input plus output size.

3 Static Traffic Flows with Congestion

In the previous section we have considered flows over time where transit times on the arcs are fixed. The latter assumption is no longer true in situations where congestion does occur. Congestion means that the transit time τ_e on an arc e is no longer constant, but a monotonically increasing, convex function $\tau_e(x_e)$ of the flow value x_e on that arc e. Congestion is inherent to car traffic, but also occurs in evacuation planning, production systems, and communication networks. In all of these applications the amount of time needed to traverse an arc of the underlying network increases as the arc becomes more congested.

Congestion in static traffic networks has been studied for a long time by traffic engineers, see e. g. [89]. It models the rush hour situation in which flow between different origins and destinations is sent over a longer period of time. The usual objective to be optimized from a global system-oriented point of view is the overall road usage, which can be viewed as the sum of all individual transit times.

3.1 User Equilibrium and System Optimum

From a macroscopic point of view, such a static traffic network with congestion can be modeled by a multicommodity flow problem, in which each commodity $i \in K$ represents two locations in the network, between which d_i "cars" are to be routed per unit of time. The data (s_i, t_i, d_i), $i \in K$, form the so-called *origin-destination matrix.* Feasible routings are given by flows x, and the "cost" $c(x)$ of flow x is the total travel time spent in the network. On arc e, the flow x then induces a transit time $\tau_e(x_e)$, which is observed by all x_e flow units using that arc e. So the total travel time may be written as $c(x) = \sum_{e \in E} x_e \, \tau_e(x_e)$.

It is usually more convenient to think of a feasible routings in terms of *flows on paths* instead of *flows on arcs.* Then, for every commodity i, the d_i amounts of flow sent between s_i and t_i are routed along certain paths P from the set $\mathcal{P}_i$ of possible paths between s_i and t_i. These paths P can be seen as the choice of routes that users take who want to drive from s_i to t_i. On the other hand, any solution x in path formulation can be seen as a route recommendation to the users of the traffic network, and thus as a route guidance strategy. Therefore, it is reasonable to study properties of flows as route recommendations and investigate their quality within this model.

Current route guidance system are very simple in this respect and usually restrict to recommend shortest or quickest paths without considering the effect on the congestion that their recommendation will have. Simulations show that users of such a simple route guidance system will—due to the congestion caused by the recommendation—experience longer transit times when the percentage of users of such a system gets larger.

Without any guidance, without capacity restrictions, but full information about the traffic situation, users will try to choose a fastest route and achieve a Nash equilibrium, a so-called *user equilibrium.* Such an equilibrium is defined by the property that no driver can get a faster path through the network when everybody else stays with his route. One can show that in such an equilibrium state all flow carrying paths P in each $\mathcal{P}_i$ have the same transit time $\tau_P = \sum_{e \in P} x_e \, \tau_e(x_e)$, and, furthermore, if all τ_e are strictly increasing and twice differentiable, then the value of the user equilibrium is unique. This is no longer true in capacitated networks [21].

So in uncapaciated networks, the user equilibrium is characterized by a certain fairness, since all users of the same origin destination pair have the same transit time. Therefore, is has been widely used as a reasonable solution to the static traffic problem with congestion. This is supported by recent results showing that, under reasonable assumption about obtaining traffic information, a simple replication-exploration strategy will indeed converge to the Nash equilibrium and the convergence (the number of rounds in which users choose new routes) is polynomial in the representation length of the transit time functions [26]. So users of a static traffic system can learn the user equilibrium efficiently.

The advantage of the user equilibrium lies in the fact that it can be achieved without traffic control (it only needs selfish users) and that it is fair to all users with the same origin and destination. However, it does not necessarily optimize the total travel time. Roughgarden and Tardos [83] investigate the difference that can occur between the user equilibrium and the system optimum, which is defined as a flow x minimizing the total travel time $\sum_{e\in E} x_e\,\tau_e(x_e)$. The ratio $c(UE)/c(SO)$ of the total travel time $c(UE)$ in the user equilibrium over that in the system optimum is referred to as *the price of anarchy.* In general, it may become arbitrarily large than in the system optimum, but it is never more than the total travel time incurred by optimally routing twice as much traffic [83]. Improved bounds on the price of anarchy can be given for special classes of transit time functions [83,86].

Another unfavorable property of the user equilibrium is a non-monotonicity property with respect to network expansion. This is illustrated by the *Braess paradox*, where adding a new road to a network with fixed demands actually *increases* the overall road usage obtained in the updated user equilibrium [14,89,39].

So one may ask what can be achieved by centralized traffic management. Of course, the system optimum would provide the best possible solution by definition. But it may have long routes for individual users and thus misses fairness, which makes it unsuited as a route guidance policy (unless routes are enforced upon users by pricing and/or central guidance). To quantify the phenomenon of long routes in the system optimum, Roughgarden [82] introduces a measure of unfairness. The *unfairness* is the maximum ratio between the transit time along a route in the system optimum and that of a route in the user equilibrium (between the same origin-destination pair). In principle, the unfairness may become arbitrarily large, but like the price of anarchy, it can be bounded by a parameter $\gamma(\mathcal{C})$ of the class $\mathcal{C}$ of underlying transit time functions [86]. For instance, $\gamma\{\text{polynomials of degree } p \text{ with nonnegative coefficients}\} = p+1$.

In computational respect, user equilibrium and system optimum are quite related problems in networks without capacities on the arcs. In fact, given convex and twice differentiable transit time functions $\tau_e(x_e)$, the system optimum is the user equilibrium for the transit time functions $\bar{\tau}_e(x_e) = \tau_e(x_e) + x_e\frac{d\tau_e(x_e)}{dx_e}$, and the system optimum is the unique user equilibrium for the transit time functions $\hat{\tau}_e(x_e) = \frac{1}{x_e}\int_0^{x_e}\tau_e(t)dt$, see [11].

So they are both instances of convex optimization with convex separable objective and linear constraints. They are usually solved with the Frank-Wolfe algorithm [36], which is essentially a feasible direction method, and several variants thereof such as the Partan algorithm [32], which performs a more intelligent line search. Computing the routes of the user equilibrium is usually referred to as *traffic assignment problem.* It was originally introduced by Wardrop [93] in order to model natural driver behavior, and has been studied extensively in the literature, see [89,78]. In the absence of arc capacities, the linearized problem decomposes into separate shortest path problems for every origin-destination pair (commodity), where the length of an arc e is given by the marginal cost $\bar{\tau}_e(x_e) = \tau_e(x_e) + x_e\frac{d\tau_e(x_e)}{dx_e}$. For capacitated networks, one has to solve a multicommodity flow problem instead.

3.2 Constrained System Optimum

Can one overcome the unfairness in the system optimum by introducing constraints on the paths that may be recommended as routes? This problem has been investigated by Jahn et al. in [51] for static traffic networks with congestion. For each commodity i they consider only a subset $\bar{\mathcal{P}}_i \subseteq \mathcal{P}_i$ of *admissible paths*. In the simplest case, admissibility is defined by geographical length in the following way. Every arc e has a geographical length ℓ_e, and path $P \in \mathcal{P}_i$ is *admissible* if its geographical length $\ell_P := \sum_{e \in P} \ell_e$ does not exceed the geographical length of a shortest path between s_i and t_i by a certain, globally controlled *fairness factor* $L > 1$ (e. g., $L = 1.2$). The corresponding optimum is a *constrained system optimum* for the original problem and the total travel time increases with $1/L$. An example from [51] with real data is given in Figure 5.

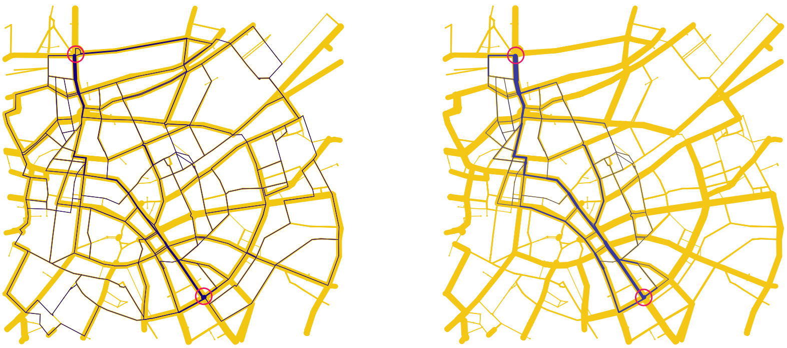


Fig. 5. System optimum with restrictions on path lengths: One commodity routed through the road network between the marked nodes. The left-hand side image displays the system optimum in which flow is distributed over the whole network in order to avoid high arc flows that would incur high arc transit times. In the right-hand side image the same amount of flow gets routed, but this time with a restriction on the geographical path lengths. Line thickness denotes arc capacity (yellow) and arc usage (blue).

More precisely, Jahn et al. introduce the concept of the *normal length* of a path, which can be either its traversal time in the uncongested network, its traversal time in user equilibrium, its geographic distance, or any other appropriate measure. The only condition imposed on the normal length of a path is that it may not depend on the actual flow on the path. Equipped with this definition, they look for a *constrained system optimum* (*CSO*) in which no path carrying positive flow between a certain origin-destination pair is allowed to exceed the normal length of a shortest path between the same origin-destination pair by more than a tolerable factor $L > 1$. By doing so, one finds solutions that are fair and efficient at the same time. This approach implements a compromise between user equilibrium and system optimum, which meets a demand expressed e. g. by

Beccaria and Bolelli [10]: The goal should be to "find the route guidance strategy which minimizes some global and community criteria with individual needs as constraints".

Computations in [51] with the street network of Berlin and networks from the *Transportation Network Test Problems* website [5] indicate that the *CSO* with transit times in the user equilibrium as normal lengths is the right concept to measure fairness. The resulting total travel time cannot exceed that of the user equilibrium, i.e., $c(SO) \leq c(CSO) \leq c(UE)$, and achieves a much better fairness than the unrestricted system optimum, while at the same time avoiding the larger road usage and non-monotonicity of the user equilibrium. An example is given in Figure 6 taken from [51].

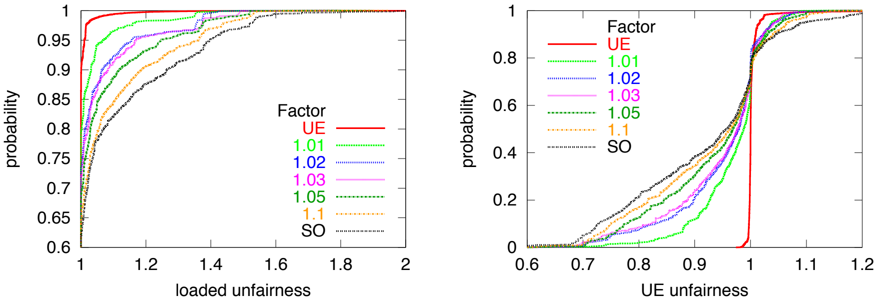


Fig. 6. Objective values and unfairness distributions for the area Neukölln in Berlin. Normal lengths are equal to transit times in user equilibrium. Total travel times are distinctively smaller than in equilibrium, while the fraction of users traveling longer than in equilibrium is substantially smaller.

Besides the unfairness with respect to the user equilibrium, Jahn et al. consider also an intrinsic unfairness of the suggested routes, which is measured as the largest ratio ℓ_{P_1}/ℓ_{P_2} of the geographical path lengths ℓ_{P_j} of two flow carrying paths P_1, P_2 for any origin-destination pair i. The intrinsic unfairness of the system optimum may go up to 3 and more, while its control via the fairness factor L also implies a strongly correlated fairness for other "length" measures such as actual transit times or transit times in the uncongested network.

3.3 Algorithmic Issues

To solve Problem *CSO*, Jahn et al. use the Partan variant of the Frank-Wolfe algorithm mentioned above. The model is path-based, where every potential path $P \in \cup_{i \in K} \bar{\mathcal{P}}_i$ is represented by a path flow variable x_P. However, these paths are only given implicitly by the length constraints $\sum_{e \in P} \ell_e \leq L\,\lambda(s_i, t_i)$ for a path between s_i and t_i, where $\lambda(s_i, t_i)$ is the normal path length between s_i and t_i. Because there may be exponentially many paths, they are only generated as needed. For that reason, the algorithm can be considered as a column generation method. This becomes even more obvious when the network is capacitated. In that case, in order to find the best descent direction, one has to solve a linear program (a multicommodity flow problem) in path variables with column generation.

One then maintains a linear program (the *master program*) with a subset of the variables (enough to contain a basis). Checking the master program for optimality then reduces to a *constrained shortest path problem* in the given network: Compute a shortest path (where the length is influenced by the dual solution of the master program) such that the normal length constraint $\sum_{e \in P} \ell_e \leq L\,\lambda(s_i, t_i)$ is preserved. Then either optimality is established and the current solution in the master problem is optimal, or there are paths that are “too long”. These paths are added to the master problem (usually in exchange for others) and one iterates.

This is the point in the algorithm where also other conditions on the paths could be considered. They can be dealt with in the current algorithm if the shortest path problem resulting from the linear programming optimality conditions can be combined with the restrictions defining the paths.

The constrained shortest path problem is NP-hard in the weak sense. There are many different approaches to solve this problem in practice. Jahn et al. implemented the label correcting algorithm of [2] because of its superior computational efficiency. The algorithm fans out from the start node s and labels each reached node $v \in V$ with labels of the form $(d_\ell(v), d_\tau(v))$. For each path from s to v that has been detected so far, $d_\tau(v)$ represents its transit time and $d_\ell(v)$ its distance. During the course of the algorithm, several labels may have to be stored for each node v, namely the Pareto-optimal labels of all paths that have reached it. This labeling algorithm can be interpreted as a special kind of branch-and-bound with a search strategy similar to breadth-first search. Starting from a certain label of v, one obtains lower bounds for the remaining paths from v to t by separately computing ordinary shortest path distances from v to t with respect to transit times τ_e and lengths ℓ_e, respectively. If one of these bounds is too large, the label can be dismissed. The algorithm reduces essentially to a repeated application of Dijkstra’s algorithm and runs polynomial in the maximum number of labels generated at a node.

The non-linear part of the algorithm is dealt with as in the Frank-Wolfe algorithm. One first computes a feasible direction $y - x = \operatorname{argmin}\{\nabla c(x)^T y \mid y \text{ feasible flow}\}$ of the objective function c at the current flow x and then a stepsize λ by line search. The next solution is then obtained as $x := x + \lambda(y - x)$. At first glance, this natural approach seems to be infeasible since the objective function $c(x)$ involves all of the many path variables x_P. But the scalar product $\nabla c(x)^T y$ can equivalently be expressed in arc variable vectors, thus reducing the number of variables to the number of arcs in the network. Also the line search can be reduced to the number of paths that actually carry flow, which does not get too large.

Figure 7 illustrates the nesting of the different steps in the algorithm. The second step (solving the linear program) is only necessary for capacitated networks. It becomes clear that algorithms for shortest paths and constrained shortest

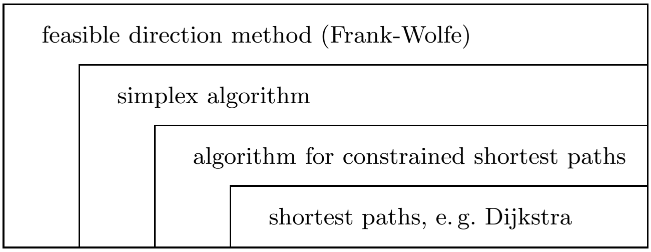

feasible direction method (Frank-Wolfe)
simplex algorithm
algorithm for constrained shortest paths
shortest paths, e. g. Dijkstra

Fig. 7. Steps in computing the constrained system optimum. The second step (solving the linear program) is only necessary for capacitated networks.

paths are at the core of the algorithm and that their run time largely determines the efficiency of the whole algorithm.

The algorithm has been tested extensively in [51]. Most notably, the time needed to compute a constrained system optimum is typically not larger than that for computing an unconstrained system optimum, and it is only somewhat larger than that for getting a user equilibrium. In fact, the problem of finding a constrained system optimum becomes computationally more costly with increasing values of the tolerance factor L. The reason is that the number of allowable paths increases. However, the constrained shortest path subproblems become easier because the normal lengths are less binding. In this trade-off situation, the total work and the number of iterations increase, but the work per iteration decreases. Generally, most of the time is spent on computing constrained shortest paths (which implies that improved algorithms for this subproblem would yield greatly improved overall performance). Instances with a few thousand nodes, arcs and commodities can be solved on an average PC within minutes. Bigger instances like Berlin take longer but can also be solved without difficulty in less than an hour.

Additional speedups are obtained by König [62] with a Lagrangian relaxation that relaxes the coupling condition between flow rates on paths and arcs, i.e., the condition that $x_e = \sum_{P:e\in P} x_P$. The problem can then be separated into commodities and determining the Lagrangian dual reduces to computing resource-constrained shortest paths and evaluation of a simple twice differentiable function and its derivatives. The actual computation of the Lagrangian dual is done with the Proximal Analytic Center Cutting Plane Method of Babonneau et al. [4]. Altogether, this leads to a speedup by a factor of 2 compared with the Partan algorithm.

3.4 Acceleration of Shortest Path Computation

Figure 7 shows that repeated (constrained) shortest path calculations form the bottleneck for computing the constrained system optimum. Similar requirements for repeated shortest path calculations arise in algorithms for navigation systems, where the network is again considered to be static (at least over certain periods).

The last few years have witnessed tremendous progress in speeding up such calculations by preprocessing the network data for more efficient repeated shortest path calculations. A good overview on this progress is given by the theses of Wilhalm [95], Schilling [85], Schultes [88] and the volume of papers of the 2006 Dimacs Challenge on Shortest Paths (in preparation) [22].

Köhler et al. [57,85] investigate a generalization of a partition-based arc labelling approach has that is referred to as the *arc-flag approach.* The basic idea of the arc-flag approach using a simple rectangular geographic partition was suggested by Lauther in [66,67] and patented in [25]. The arc-flag approach divides the graph $G = (V, A)$ into regions $r \in R$ and gathers information for each arc $a \in A$ and for each region $r \in R$ ($V = \bigcup_{r \in R} r$) on whether the arc a is on at least one shortest path leading into region r. For each arc $a \in A$ this information is stored in a flag (bit) vector f_a. The vector f_a contains a flag (`true` or `false`) for each region $r \in R$ indicating whether the arc a can contribute to answering shortest path queries for nodes in region r or not, see Figure 8. Thus, the size of each flag vector is determined by the number $|R|$ of regions and the number of flag vectors is bounded by the number $|A|$ of arcs. Since the actual number of unique flag vectors can be much smaller than the number of arcs, storing the flag vectors at one point and adding an index (or pointer) to each arc can reduce the extra amount of memory below the obvious $|A||R|$ bits. The number of regions depends on the input graph size, but can be kept to a moderate size: 200 regions already lead to considerable speedups on instances with 24M nodes and 58M edges.

The arc-flags are used in a slightly modified Dijkstra computation to avoid exploring unnecessary paths. This means that one checks the flag entry of the corresponding target region (the region where the target node t belongs to) each time before the Dijkstra algorithm tries to traverse an arc. Thus, implementing the arc-flags is one of the simplest acceleration modifications of the standard Dijkstra algorithm and therefore suggests itself for straightforward usage in existing code bases.

The choice of the underlying partition is crucial for the speedup of the arc-flag acceleration of Dijkstra's algorithm. In [57] a multi-way arc separator is suggested

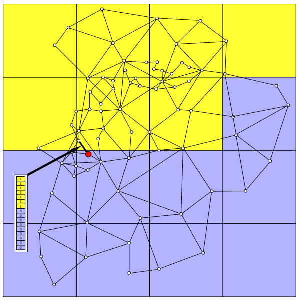


Fig. 8. Rectangular decomposition. Every arc $e = (u, v)$ carries a bit vector with a bit for every region r. The bit of region r is set to 1 iff e is on a shortest path from u to a node in r.

as an appropriate partition for the arc-flags, see Figure 9 for an example. This improvement achieved much better speedups compared to the original arc-flag version in [67]. For instance, using this one can reach acceleration factors 10 times higher than with Lauther's version of the arc-flags (on networks with up to 0.3M nodes, 0.5M arcs and 278 bits of additional information per arc). This study was further refined by Möhring et al. [76].

When combining the arc-flags with a multi-way arc separator partition and a bi-directed search, the overall performance of the method is competitive to those of other acceleration techniques such as the highway hierarchy method of Sanders and Schultes [84] but requires much more preprocessing time. Hilger et al. [58] reduce the preprocessing times significantly and also improve the efficiency of queries and the space requirements. This is achieved by a new *centralized shortest path* algorithm which computes distances simultaneously from a set of starting vertices instead of one starting vertex. Another improvement on the preprocessing time is achieved by removing small attached structures for which the arc-flags can be calculated in a much easier way. Note that this reduction is only performed during preprocessing, queries are calculated on the unmodified graph using the pre-calculated information.

On continental road networks like the US network (24M nodes, 54M edges) this method only needs a few hours to complete the pre-calculation. This approach is the first to apply the arc-flag accelerated shortest path method to networks of this size. Recently, Bauer and al. [6] have demonstrated that the arc flag method currently constitutes the best performing purely goal-directed acceleration technique and that the preprocessing can even be improved further.

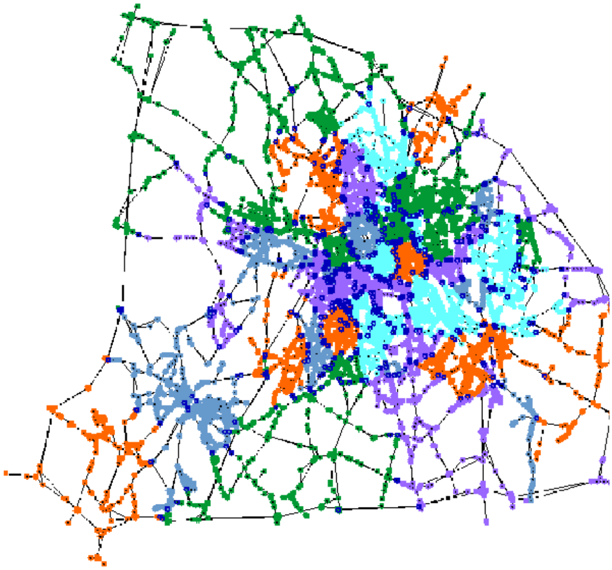


Fig. 9. Berlin divided by a hierarchical separator approach

If preprocessing is not possible, e.g. because the arc values change too often (which is the case in the algorithms for the restricted user optimum), one can still apply goal directed and bidirected search for shortest paths. Here the main underlying idea is to distort arc weights in such a way, that Dijkstra's algorithm prefers to explore nodes that are closer to the target node, thus reducing the time it takes to reach the source. Figure 10 illustrates the areas of the graph that are explored in the respective searches.

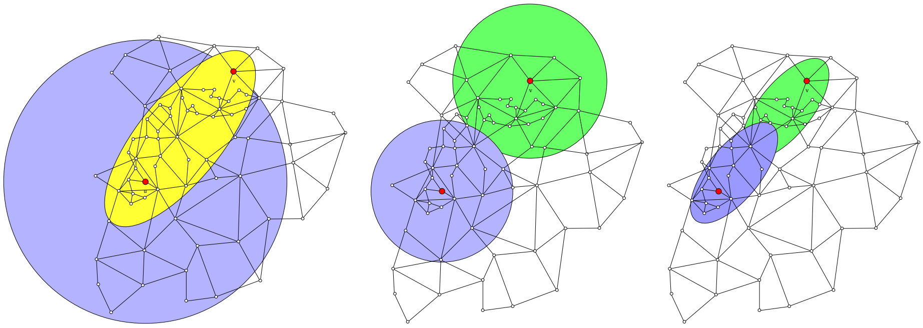


Fig. 10. Variants of simple search speedup techniques: goal directed, bidirected, and their combination (from left to right)

The goal directed approaches can also be applied to some extent to the constrained shortest path problem [57]. Here, the simple goal oriented search showed the best speedup, as the bidirectional variant suffers from the lack of a good stopping criterion.

4 Flows over Time with Flow Dependent Transit Times

In the previous sections we particularly considered flow models that either were able to capture flow variations with respect to time (Section 2.2), or transit time variations with respect to the flow situation on an arc (Section 3). Both aspects are important features that have to be taken into account when optimizing traffic systems. However, while in the previous sections we handled these features separately, we will now investigate what can be done if we have to deal with both of them at the same time. To be more precise, we study flow variations over time in networks where transit times on the arcs vary with the amount of flow on that particular arc. While both static flow theory with flow dependent transit times and the theory of flows over time with constant transit times are well studied areas of research, the field of flows over time with flow dependent transit times has attracted attention only in the last couple of years. One main reason for this seems to be the lack of well defined models for this kind of flows, which is due to the more complicated setting. Standard approaches from flows over time with constant transit times seem not easily be carried over to this case and we will explain some of the occurring problems in the following. There is, in fact, a variety of different approaches to define an applicable model. Unfortunately, all of them are capturing only some but not all features of flows over time with flow dependent transit times. However, they hopefully will be helpful for the design of useful models and algorithms in the future.

4.1 Problems of Flow Dependent Transit Times

One significant property of flows over time in networks is that, in contrast to static flow problems, flow values on arcs may change with time. As mentioned before, in many applications one has to deal also with the phenomenon that the time taken to traverse an arc varies with the current flow situation on this arc. It is a highly nontrivial problem to map these two aspects into an appropriate and tractable mathematical network flow model and there are hardly any algorithmic techniques known which are capable of providing reasonable solutions even for networks of rather modest size. The crucial parameter for modeling temporal dynamics of time-dependent flows is the presumed dependency of the actual transit time τ_e on the current (and maybe also past) flow situation on arc e. Unfortunately, there is a tradeoff between the need of modeling this usually highly complex correlation as realistically as possible and the requirement of retaining tractability of the resulting mathematical program.

As for the static case, a fully realistic model of flow-dependent transit times on arcs must take density, speed, and flow rate evolving along the arc into consideration [38]. Unfortunately, even the solution of mathematical programs relying on simplifying assumptions is in general still impracticable, i. e., beyond the means of state-of-the-art computers, for problem instances of realistic size (as those occurring in real-world applications such as road traffic control).

When considering the correspondence between transit time and flow on a particular arc, the first question in a time dependent model is, of course, which flow or flow rate (flow units traversing an arc per time unit) do we measure? When deciding about the transit time of a given unit of flow in an arc e at a particular point in time t, what flow rate value do we consider to determine this transit time? One possibility would be, to take the inflow rate, i. e., the flow rate at the beginning of the arc e at the moment that the particular flow unit enters e. When looking at this approach in the setting of traffic networks, one realizes its drawbacks. Even if the number of cars which currently enter a street is small compared to the capacity of the street, their transit times might nevertheless be huge due to traffic congestion caused by a large number of cars which have entered the arc earlier. Another plausible choice would be to consider the whole amount of flow that is on arc e at time t, i. e., the load of e at time t. Yet, this has its disadvantages as well, since when considering all cars on a street e at some point in time t, one even counts those cars that are behind the particular car of interest.

Besides the question how to measure the flow situation on an arc, an important figure in the search for a good model is the transit time function itself. Due to the inherent complexity of the time- and flow-dependent setting, models in the literature often rely on relatively simple transit time functions, based on those, used in static models of flow-dependent networks, e. g., the Davidson function or the function of the BPR (see [89]). While these functions $\tau_e(x_e)$ are given for a static flow value x_e, that is not varying over time, one can interpret $\tau_e(x_e)$ as the transit time on arc e for the static flow rate x_e. Of course, these functions mean a simplification of the actual flow situation in a time- and flow-dependent model. However, their use is justified already since they are a generalization of the well-established models for the static case and thus often allow to compare the quality of a computed solutions to the solutions of the static model.

When developing flow models for real-world applications, a useful model often has to satisfy certain additional requirements, which sometimes increase the complexity of this model enormously. One prominent such requirement that is present especially in traffic applications is the so-called *FIFO* or *first-in-first-out* property. This property states that no unit of flow A entering an arc e after some flow unit B exits this arc before B; in other words, overtaking of flow units is not permitted. While a violation of this causes no problems for traffic when considering single cars, it is considered not permitted by traffic scientists for large groups of cars. In a model with constant transit time this requirement is trivially satisfied since all flow through an arc travels with the same travel time. However, in a model with variable transit times on the arcs it is often very hard to guarantee.

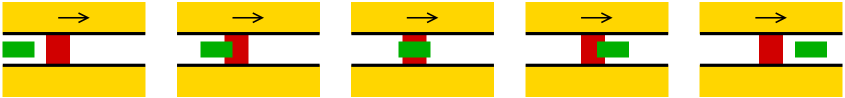


Fig. 11. Violations of the first-in-first-out property

4.2 Models and Algorithms

In the following we discuss some approaches that can be found in the literature. For a more detailed account and further references we refer to [3,69,79,81].

One of the first models for time dependent flows with flow dependent transit times has been defined by Merchant and Nemhauser [73]. They formulate a non-linear and non-convex program in which time is discretized. In their model, the outflow out of an arc in each time period solely depends on the amount of flow on that arc at the beginning of the time period. Instead of using a transit time function, as discussed above, they use for each arc e a cost function $h_{e,t}$ and an exit function g_e, where $h_{e,t}$ accounts for the transit time that is spend by the flow on arc e in period t. Now, if x is the amount of traffic on an arc e at the beginning of time period t, a cost $h_{e,t}(x)$ is incurred and during period t an amount of traffic $g_e(x)$ exits from arc e. In their setting Merchant and Nemhauser were able to consider networks of general topology and multiple origins, but only a single sink. Although this model seems to be not easily applicable to real-world applications, it has been an important stepping stone in the study of time- and flow-dependent models. Still, the non-convexity of this model causes both analytical and computational problems.

Later, Merchant and Nemhauser [74] and Carey [17] further studied this model and described special constraint qualifications which are necessary to guarantee optimality of a solution in this model. Carey [18] introduces a slight revision of the model of Merchant and Nemhauser which transforms the non-convex problem into a convex one.

A rather different approach to modeling flows over time with flow dependent transit times was used by Carey and Subrahmanian [19]. Before we look at their model in more detail, we shortly discuss the problems occurring with the time expanded network approach of Section 2.2.

Recall that the main advantage of the time expanded network is that we can apply known algorithms for static networks to solve the corresponding time dependent problems. Unfortunately, this simple approach does not easily carry over to the case of flow dependent transit times. Consider a simple network with only two arcs. Initially, the flow value on each arc is zero and, as each arc has a certain empty transit time (in our example the transit time of each arc is one time unit), we can easily construct a time expanded network, as described earlier (see Figure 12). Again, we can consider a path P from some source s to a sink t in the original network and find the corresponding path in the time-expanded version that captures the transit times of the different arcs. However, if we now send some flow on this path P through the network, then the problem of this approach becomes evident: since the transit times of the arcs vary with the amount of flow on them, the time expanded graph changes its adjacencies and its structure. In fact, the arcs of the path corresponding to P in the original network now become incident to completely different copies of the vertices in the time-expanded network. As a consequence we cannot apply standard algorithms for static flow problems any more since they rely heavily on the fact that the underlying graph is not changing. Thus, the advantage of the time-expanded approach seems to be lost.

In spite of this daunting observation, Carey and Subrahmanian [19] were able to define a variant of the time expanded network that captures at least some features of flow dependent transit times. Their network again has fixed transit times on the arcs. But for each time period, there are several copies of an arc of the underlying 'static' network corresponding to different transit times. Now changing flow values on the arcs is to result no longer in a change of adjacencies in the graph but instead different copies of the same arc should be used for different amounts of flow. In order to enforce these flow-dependent transit times, special capacity constraints are introduced which give rise to a dependency between the flow on all copies of an arc corresponding to one time step. As a consequence of these generalized capacity constraints, the resulting static problem on the modified time-expanded graph can no longer be solved by standard network flow techniques but requires a general linear programming solver. This constitutes a serious drawback with regard to the practical efficiency and applicability of this model.

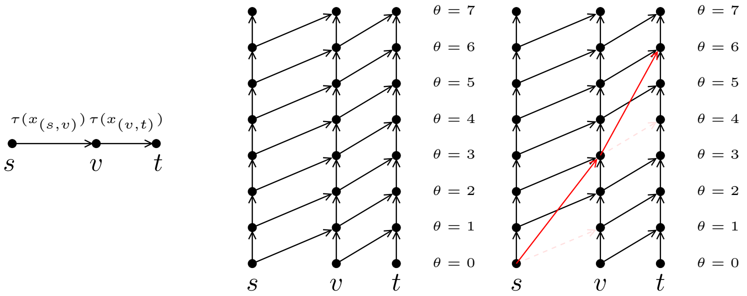


Fig. 12. A simple two-arc network together with a time-expansion for empty transit times, and the time-expansion when one unit of flow is sent from s to t, showing the problems of a time expansion for flow dependent transit times

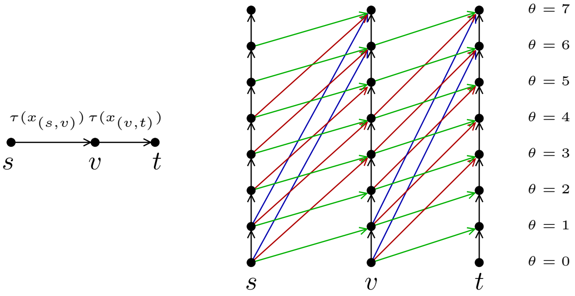


Fig. 13. Idea of expanded model of Carey and Subrahmanian; here for a simple two-arc network with transit time function $\tau(x_e)$, causing transit times 1, 3, and 6, depending on the amount of flow x_e entering arc e

Although the models mentioned up to this point mark some important developments in the study of flows over time with flow dependent transit times, they have a drawback in common; they are not suitable for efficient optimization methods. In contrast to the models for static flows and for flows over time with constant transit times, here fast standard flow algorithms cannot be applied. This is of course partially due to the complexity of these models. However, for solving real problems, also on large networks one has to have fast algorithmic tools.

The following model was suggested by KÃűhler and Skutella [60,61]. It is inspired by the earlier mentioned results of Ford and Fulkerson and of Fleischer and Skutella (see Section 2.2). Although the result of Ford and Fulkerson for computing a maximum flow over time for a fixed time horizon T cannot be generalized to the more general setting of flow-dependent transit times, it is shown in [60,61] that there exists at least a provably good temporally repeated flow for the quickest flow problem, i. e., for the problem of sending a certain amount of flow from some vertex s through the network such that it reaches the sink t as fast as possible.

As for the case of fixed transit times, and in contrast to the before mentioned models for flow dependent transit times, this flow can be determined very efficiently. Since the transit times are no longer fixed, the linear min-cost flow problem considered by Ford and Fulkerson now turns into a *convex cost* flow problem. Under very mild assumptions on the transit time functions τ_e, the resulting optimal static flow can be turned into a temporally repeated flow which needs at most twice as long as a quickest flow over time.

This result is based on the fairly general model of flow-dependent transit times where, at each point in time, the speed on an arc depends only on the amount of flow (or *load*) which is currently on that arc. This assumption captures for example the behavior of road traffic when an arc corresponds to a rather short street (notice that longer streets can be replaced by a series of short streets).

The next model that we present here is again based on the time-expanded network, known from flows over time with constant transit time. As in the model of Carey and Subrahmanian [19], KÃűhler, Langkau and Skutella [56,65] suggest a generalized time-expanded network with multiple copies of each arc for each time step. However, in contrast to the model in [19], additional 'regulating' arcs are introduced which enable to enforce flow-dependent transit times without using generalized capacity constraints. As a result, one can apply the whole algorithmic toolbox developed for static network flows to this generalized time-expanded network, also called the *fan graph* (see Figure 14).

The underlying assumption for this approach (as for the related approach of Carey and Subrahmanian [19]) is that at any moment of time the transit time on an arc solely depends on the current rate of inflow into that arc. We therefore speak of *flows over time with inflow-dependent transit times*, emphasizing the fact that transit times are considered as functions of the rate of inflow. Thus, in contrast to the load-dependent model developed by KÃűhler and Skutella [60,61], the flow units traveling on the same arc at the same time do not necessarily experience the same pace, as the transit time and thus the pace of every unit of flow is determined when entering the arc and remains fixed throughout.

As in the case of constant transit times, a drawback of this rather general time-expanded model is the size of the constructed graph. Depending on the considered time horizon and the size of a single time step, the graph grows

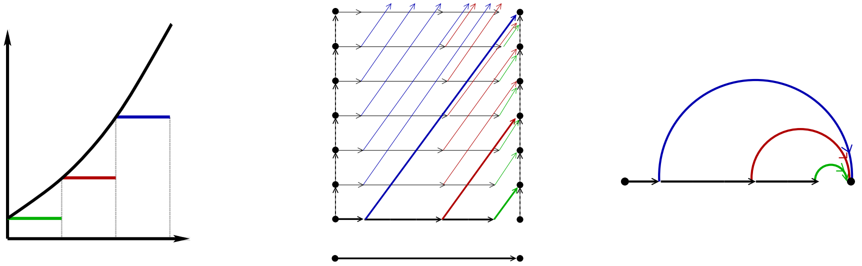


Fig. 14. Transit time function, fan-graph, and bow-graph of a single arc

rapidly. Hence, when considering large instances of networks, this model is again not applicable. However, as shown by KÃűhler, Langkau and Skutella [65,56] as well, there is also a time condensed version of this network, the so-called *bow-graph* of the given network. The idea is here to construct a graph with constant transit times on the arcs, such that the time expansion of this graph is again the fan-graph. Another way of looking at this condensed graph is, to see it as an expansion of the transit time function of the original graph, since now for each arc we have copies for each possible transit time of this arc (instead of having a copy for each time step in the time expanded graph). Again a set of regulation arcs guarantees that only a restricted amount of flow can transit an arc using a particular copy of this arc in the bow-graph. This bow-graph itself does not capture the time varying behavior of the flow over time. However, the interesting property here is that the bow-graph relates to the fan-graph in this model, as the original graph relates to the time expanded graph in the constant travel time model of Ford and Fulkerson. As shown in [65,56], this relationship can be exploited for developing a 2-approximation algorithm for the quickest flow problem in this model for flows over time with inflow-dependent transit times.

For the load-dependent model of KÃűhler and Skutella [60,61] the quickest flow problem can be shown to be APX-hard even for the case of only one source and one sink, implying that there is no polynomial-time approximation scheme unless P=NP. The inflow-dependent model is easier to approximate. Hall, Langkau and Skutella [65,42,43] combined the inflow-dependent model with the idea of condensed time-expanded networks (see Section 2.2) to design a fully polynomial time approximation scheme for the quickest multicommodity flow problem with inflow-dependent transit times.

Not only quickest flows but also earliest arrival flows have been studied in the context of flow dependent transit times. Baumann and KÃűhler [7,8] show that for the case of flow-dependent transit times earliest arrival flows do not always exist. However, an optimization version of the earliest arrival problem is introduced. This is done as follows. Find the minimum α such that there is a flow over time f that sends for each $\theta \in [0, T)$ at least as much flow into the sink t as can be sent into t up to time $\frac{\theta}{\alpha}$ by a maximum flow over time $f_{\max}(\frac{\theta}{\alpha})$ within this time horizon, i.e., $\text{value}(f, \theta) \geq \text{value}(f_{\max}(\frac{\theta}{\alpha}), \frac{\theta}{\alpha})$. Such a flow will be called an *α-earliest arrival flow.* Using any of the approximation algorithms for the quickest flow problem both for load-dependent and inflow-dependent flows one can design approximation algorithms for the α-earliest arrival flow problem.

Finally, we would like to mention the *rate-dependent* model. It was introduced by Hall and Schilling [44] and tries to overcome some of the deficiencies of the flow-dependent models outlined above. In particular one can avoid FIFO-violations (see Section 4.1) which might occur in the inflow-dependent model and undesired overtaking (see Section 4.1) that might occur in the load-dependent model. In the rate-dependent model the relationship between the flow-rate on an edge and the *pace* (which is defined to be the inverse of the velocity) is the central notion. Roughly speaking, the novel idea of this model is not to set a fixed pace for each possible flow rate (as done in the earlier models), but rather to allow a whole interval of pace–flow rate combinations. Hence, one ends up with a less restrictive and more realistic model. Unfortunately, these advantages are obtained on the cost of efficiency: Similarly, as for the earlier models, computing a quickest flow is NP-hard. But, in contrast, no efficient approximation algorithm, e.g. for the quickest flow in this model is known. Yet, the authors show the practical relevance of their approach by presenting a heuristic algorithm for the quickest flow problem and comparing its solutions with the solutions of the inflow-dependent model.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows. Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs (1993)
2. Aneja, Y.P., Aggarwal, V., Nair, K.P.K.: Shortest chain subject to side constraints. Networks 13, 295–302 (1983)
3. Aronson, J.E.: A survey of dynamic network flows. Annals of Operations Research 20, 1–66 (1989)
4. Babonneau, F., du Merle, O., Vial, J.-P.: Solving large-scale linear multicommodity flow problems with an active set strategy and proximal-accpm. Operations Research 54(1), 184–197 (2006)
5. Bar-Gera, H.: Transportation network test problems (2002), `http://www.bgu.ac.il/~bargera/tntp/`
6. Bauer, R., Delling, D., Sanders, P., Schieferdecker, D., Schultes, D., Wagner, D.: Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm (submitted) (2008)
7. Baumann, N.: Evacuation by Earliest Arrival Flows. Ph.D thesis, Universität Dortmund (2007)
8. Baumann, N., Köhler, E.: Approximating earliest arrival flows with flow-dependent transit times. Discrete Appl. Math. 155, 161–171 (2007)
9. Baumann, N., Skutella, M.: Solving evacuation problems efficiently: Earliest arrival flows with multiple sources. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, Berkeley, CA, pp. 399–408 (2006)
10. Beccaria, G., Bolelli, A.: Modelling and assessment of dynamic route guidance: the MARGOT project. In: Vehicle Navigation & Information Systems Conference Proceedings (VNIS 1992), pp. 117–126. IEEE, Los Alamitos (1992)
11. Beckmann, M., McGuire, C.B., Winston, C.B.: Studies in the economics of transportation. Yale University Press, New Haven (1956)
12. Ben-Akiva, M., Koutsopoulos, H., Mishalani, R., Yang, Q.: Simulation laboratory for evaluating dynamictraffic management systems. ASCE Journal of Transportation Engineering 123(4), 283–289 (1997)
13. Berlin, G.N.: The use of directed routes for assessing escape potential. National Fire Protection Association, Boston (1979)
14. Braess, D.: Über ein paradoxon aus der Verkehrsplanung. Unternehmensforschung 12, 258–268 (1968)
15. Burkard, R.E., Dlaska, K., Klinz, B.: The quickest flow problem. ZOR — Methods and Models of Operations Research 37, 31–58 (1993)
16. Busaker, R.G., Gowen, P.J.: A procedure for determining minimal-cost network flow patterns. Technical Report 15, Operational Research Office. John Hopkins University, Baltimore, MD (1961)

17. Carey, M.: A constraint qualification for a dynamic traffic assignment model. Transp. Science 20, 55–58 (1986)
18. Carey, M.: Optimal time-varying flows on congested networks. OR 35, 58–69 (1987)
19. Carey, M., Subrahmanian, E.: An approach for modelling time-varying flows on congested networks. Transportation Research B 34, 157–183 (2000)
20. Chalmet, L.G., Francis, R.L., Saunders, P.B.: Network models for building evacuation. Management Science 28, 86–105 (1982)
21. Correa, J.R., Schulz, A.S., Stier Moses, N.E.: Selfish routing in capacitated networks. Mathematics of Operations Research 29(4), 961–976 (2004)
22. DIMACS. 9th Implementation Challenge – Shortest Paths (2006)
23. Dirkse, S.P., Ferris, M.C.: Traffic modeling and variational inequalities using GAMS. In: Toint, P.L., Labbe, M., Tanczos, K., Laporte, G. (eds.) Operations Research and Decision Aid Methodologies in Traffic and Transportation Management, vol. 166, pp. 136–163. Springer, Berlin (1998)
24. Elias, P., Feinstein, A., Shannon, C.E.: Note on maximum flow through a network. IRE Transactions on Information Theory IT-2, 117–119 (1956)
25. Enders, R., Lauther, U.: Method and device for computer assisted graph processing (May 1999), `http://gauss.ffii.org/PatentView/EP1027578`
26. Fischer, S., Räcke, H., Vöcking, B.: Fast convergence to wardrop equilibria by adaptive sampling methods. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, pp. 653–662 (2006)
27. Fleischer, L., Skutella, M.: The quickest multicommodity flow problem. In: Cook, W.J., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 36–53. Springer, Heidelberg (2002)
28. Fleischer, L., Skutella, M.: Minimum cost flows over time without intermediate storage. In: Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms, Baltimore, MD, pp. 66–75 (2003)
29. Fleischer, L., Skutella, M.: Quickest flows over time. SIAM Journal on Computing 36, 1600–1630 (2007)
30. Fleischer, L.K.: Faster algorithms for the quickest transshipment problem. SIAM Journal on Optimization 12, 18–35 (2001)
31. Fleischer, L.K., Tardos, É.: Efficient continuous-time dynamic network flow algorithms. Operations Research Letters 23, 71–80 (1998)
32. Florian, M., Guélat, J., Spiess, H.: An efficient implementation of the "Partan" variant of the linear approximation method for the network equilibrium problem. Networks 17, 319–339 (1987)
33. Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. Canadian Journal of Mathematics 8, 399–404 (1956)
34. Ford, L.R., Fulkerson, D.R.: Constructing maximal dynamic flows from static flows. Operations Research 6, 419–433 (1958)
35. Ford, L.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press, Princeton (1962)
36. Frank, M., Wolfe, P.: An algorithm for quadratic programming. Naval Research Logistics Quaterly 3, 95–110 (1956)
37. Gale, D.: Transient flows in networks. Michigan Mathematical Journal 6, 59–63 (1959)
38. Gartner, N., Messer, C.J., Rathi, A.K.: Traffic flow theory: A state of the art report (1997), `http://www-cta.ornl.gov/cta/research/trb/tft.html`
39. Hagstrom, J.N., Abrams, R.A.: Characterizing braess's paradox for traffic networks. In: Proceedings of IEEE 2001 Conference on Intelligent Transportation Systems, pp. 837–842 (2001)

40. Hajek, B., Ogier, R.G.: Optimal dynamic routing in communication networks with continuous traffic. Networks 14, 457–487 (1984)
41. Hall, A., Hippler, S., Skutella, M.: Multicommodity flows over time: Efficient algorithms and complexity. Theoretical Computer Science 379, 387–404 (2007)
42. Hall, A., Langkau, K., Skutella, M.: An FPTAS for quickest multicommodity flows with inflow-dependent transit times. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) RANDOM 2003 and APPROX 2003. LNCS, vol. 2764, pp. 71–82. Springer, Heidelberg (2003)
43. Hall, A., Langkau, K., Skutella, M.: An FPTAS for quickest multicommodity flows with inflow-dependent transit times. Algorithmica 47(3), 299–321 (2007)
44. Hall, A., Schilling, H.: Flows over time: Towards a more realistic and computationally tractable model. In: Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX 2005), Vancouver, Canada, pp. 55–67. SIAM, Philadelphia (2005)
45. Hamacher, H.W., Tifecki, S.: On the use of lexicographic min cost flows in evacuation modeling. Naval Research Logistics 34, 487–503 (1987)
46. Hitchcock, F.L.: The distribution of a product from several sources to numerous localities. Journal of Mathematics and Physics 20, 224–230 (1941)
47. Hoppe, B.: Efficient dynamic network flow algorithms. Ph.D thesis, Cornell University (1995)
48. Hoppe, B., Tardos, É.: Polynomial time algorithms for some evacuation problems. In: Proceedings of the 5th Annual ACM–SIAM Symposium on Discrete Algorithms, Arlington, VA, pp. 433–441 (1994)
49. Hoppe, B., Tardos, É.: The quickest transshipment problem. Mathematics of Operations Research 25, 36–62 (2000)
50. Iri, M.: A new method of solving transportation-network problems. Journal of the Operations Research Society of Japan 26, 27–87 (1960)
51. Jahn, O., Möhring, R.H., Schulz, A.S., Stier Moses, N.E.: System-optimal routing of traffic flows with user constraints in networks with congestion. Oper. Res. 53(4), 600–616 (2005)
52. Jarvis, J.J., Ratliff, H.D.: Some equivalent objectives for dynamic network flow problems. Management Science 28, 106–108 (1982)
53. Jewel, P.A.: Optimal flow through networks. Technical Report 8, Operations Research Center. MIT, Cambridge (1958)
54. Kantorovich, L.V.: On the translocation of masses. Comptes Rendus (Doklady) de l'Académie des Sciences de l'U.R.S.S. 37, 199–201 (1942)
55. Klinz, B., Woeginger, G.J.: Minimum cost dynamic flows: The series-parallel case. In: Balas, E., Clausen, J. (eds.) IPCO 1995. LNCS, vol. 920, pp. 329–343. Springer, Heidelberg (1995)
56. Köhler, E., Langkau, K., Skutella, M.: Time-expanded graphs with flow-dependent transit times. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 599–611. Springer, Heidelberg (2002)
57. Köhler, E., Möhring, R.H., Schilling, H.: Acceleration of shortest path and constrained shortest path computation. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 126–138. Springer, Heidelberg (2005)
58. Köhler, E., Möhring, R.H., Schilling, H.: Fast point-to-point shortest path computations with arc-flags (February 2008); submitted to the Special Issue about the 9th DIMACS Implementation Challenge Workshop
59. Köhler, E., Möhring, R.H., Skutella, M.: Traffic networks and flows over time. In: Kramer, J. (ed.) DFG Research Center: Mathematics for Key Technologies, pp. 49–70. Berliner Mathematische Gesellschaft (2002)

60. Köhler, E., Skutella, M.: Flows over time with load-dependent transit times. In: Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms, San Francisco, CA, pp. 174–183 (2002)
61. Köhler, E., Skutella, M.: Flows over time with load-dependent transit times. SIAM Journal on Optimization 15, 1185–1202 (2005)
62. König, F.: Traffic optimization under route constraints with Lagrangian relaxation and cutting plane methods. In: Operations Research Proceedings 2006, pp. 53–59. Springer, Heidelberg (2007)
63. Koopmans, T.C.: Optimum utilization of the transportation system. Econometrica 17, 136–146 (1949)
64. Kotnyek, B.: An annotated overview of dynamic network flows. Rapport de recherche 4936, INRIA Sophia Antipolis (2003)
65. Langkau, K.: Flows Over Time with Flow-Dependent Transit Times. Ph.D thesis, TU Berlin (2003)
66. Lauther, U.: Slow preprocessing of graphs for extremely fast shortest path calculations. In: Lecture at the Workshop on Computational Integer Programming at ZIB (no documentation available) (1997)
67. Lauther, U.: An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In: Raubal, M., Sliwinski, A., Kuhn, W. (eds.) Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung, Münster, Germany. IfGI prints, vol. 22, pp. 219–230. Institut für Geoinformatik, Westfälische Wilhelms-Universität (2004)
68. Lovetskii, S.E., Melamed, I.I.: Dynamic network flows. Automation and Remote Control 48, 1417–1434 (1987); Translated from Avtomatika i Telemekhanika 11, 7–29 (1987)
69. Mahmassani, H.S., Peeta, S.: System optimal dynamic assignment for electronic route guidance in a congested traffic network. In: Gartner, N.H., Improta, G. (eds.) Urban Traffic Networks. Dynamic Flow Modelling and Control, pp. 3–37. Springer, Berlin (1995)
70. Martens, M., Skutella, M.: Length-bounded and dynamic k-splittable flows. In: Haasis, H.-D., Kopfer, H., Schönberger, J. (eds.) Operations Research Proceedings 2005, pp. 297–302. Springer, Heidelberg (2006)
71. Megiddo, N.: Combinatorial optimization with rational objective functions. Mathematics of Operations Research 4, 414–424 (1979)
72. Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms. Journal of the ACM 30, 852–865 (1983)
73. Merchant, D.K., Nemhauser, G.L.: A model and an algorithm for the dynamic traffic assignment problems. Transp. Science 12, 183–199 (1978)
74. Merchant, D.K., Nemhauser, G.L.: Optimality conditions for a dynamic traffic assignment model. Transp. Science 12, 200–207 (1978)
75. Minieka, E.: Maximal, lexicographic, and dynamic network flows. Operations Research 21, 517–527 (1973)
76. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning graphs to speed-up Dijkstra's algorithm. ACM Journal of Experimental Algorithmics 11, Article No. 2.8 (2006)
77. Nagel, K., Esser, J., Rickert, M.: Large-scale traffic simulations for transportation planning. In: Stauffer, D. (ed.) Annual Review of Computational Physics VII, pp. 151–202. World Scientific Publishing Company, Singapore (2000)
78. Patriksson, M.: Traffic Assignment Problems: Models and Methods. VSP International Science Publishers, Utrecht (1994)

79. Powell, W.B., Jaillet, P., Odoni, A.: Stochastic and dynamic networks and routing. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (eds.) Network Routing. Handbooks in Operations Research and Management Science, ch. 3, vol. 8, pp. 141–295. North-Holland, Amsterdam (1995)
80. Prigogine, I., Herman, R.: Kinetic Theory of Vehicular Traffic. Elsevier Science B. V., Amsterdam (1971)
81. Ran, B., Boyce, D.E.: Modelling Dynamic Transportation Networks. Springer, Berlin (1996)
82. Roughgarden, T.: How unfair is optimal routing? In: Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms, San Francisco, CA, pp. 203–204 (2002)
83. Roughgarden, T., Tardos, É.: How bad is selfish routing? In. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science FOCS 2000, pp. 93–102 (2000)
84. Sanders, P., Schultes, D.: Highway hierarchies hasten exact shortest path queries. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 568–579. Springer, Heidelberg (2005)
85. Schilling, H.: Route Assignment Problems in Large Networks. Ph.D thesis, Technische Universität Berlin (2006)
86. Schluz, A.S., Stier Moses, N.E.: Efficiency and fairness of system-optimal routing with user constraints. Networks 48(4), 223–234 (2006)
87. Schrijver, A.: On the history of the transportation and maximum flow problem. Mathematical Programming 91, 437–445 (2002)
88. Schultes, D.: Route Planning in Road Networks. Ph.D thesis, Universität Karlsruhe, TH (2008)
89. Sheffi, Y.: Urban Transportation Networks. Prentice Hall, Englewood Cliffs (1985), `http://web.mit.edu/sheffi/www/urbanTransportation.html`
90. Skutella, M.: An introduction to network flows over time. In: Cook, W., Lovász, L., Vygen, J. (eds.) Research Trends in Combinatorial Optimization, pp. 451–482. Springer, Berlin (2009)
91. Tjandra, S.: Dynamic Network Optimization with Application to the Evacuation Problem. Ph.D thesis, Univeristät Kaiserslautern. Shaker Verlag, Aachen (2003)
92. Verkehrsmanagement — Eine Bestandsaufnahme. Broschüre. Erstellt von Lösch & Partner GmbH, München, in Zusammenarbeit mit diversen deutschen Automobilherstellern (1995)
93. Wardrop, J.G.: Some theoretical aspects of road traffic research. Proceedings of the Institution of Civil Engineers 1(2), 325–362 (1952)
94. Wilkinson, W.L.: An algorithm for universal maximal dynamic flows in a network. Operations Research 19, 1602–1612 (1971)
95. Willhalm, T.: Engineering Shortest Paths and Layout Algorithms for Large Graphs. Ph.D thesis, Universität Karlsruhe (TH), Karlsruhe (2005)

Interactive Communication, Diagnosis and Error Control in Networks

Rudolf Ahlswede[1] and Harout Aydinian[2,⋆]

[1] University of Bielefeld, Department of Mathematics, POB 100131,
D-33501 Bielefeld, Germany
ahlswede@math.uni-bielefeld.de
[2] ayd@math.uni-bielefeld.de

Abstract. In this survey we present our results on the subjects: Connectors in communication networks, Fault diagnosis in large multiprocessor networks, Unconventional error-correcting codes, Parallel error-control codes performed in the project "Algorithmik großer und komplexer Netzwerke".

Further work partially done there is commented upon in a last section.

1 Introduction

The study of **Connectors in Communication Networks** started with pioneering works by Shannon [47], Slepian [48], and Clos [23], in connection with practical problems in designing switching networks for telephone traffic. Later they were also studied as useful architectures for parallel machines (see [41]). Connectors are also related to *expander graphs*, a rapidly developing subject in the last three decades, which have found extensive applications (see [30]) in computer science, error correcting codes, cryptography etc.

An (n, N, d)–connector is an acyclic digraph with n inputs and N outputs in which for any injective mapping of input vertices into output vertices there exist n vertex-disjoint paths of length d joining each input to its corresponding output. Our goal is construction of sparse $(n, N, 2)$–connectors (depth 2 connectors) when $n \ll N$. Such connectors are of particular interest in the design of sparse electronic switches. Also they may be useful as building blocks in multistage connectors.

The probabilistic argument in Baltz *et al.* [16] shows the existence of $(n, N, 2)$–connectors of size (number of edges) $O(N)$, if $n \leq N^{1/2-\varepsilon}$, $\varepsilon > 0$.

Our main results stated in Section 2 are explicit constructions of connectors. For integers $t > 2$, $n \geq t^t$, $N = \Omega(n^t)$ our construction gives $(n, N, 2)$-connectors of size $Nn^{\frac{1}{t}(1+o(1))}$. In particular, for all n and $N > N(n)$ this gives connectors of size $2N \log n/(1 + o(1)) \log \log n$.

⋆ The author supported by DFG-Schwerpunkt Nr.1126 "Algorithmik großer und komplexer Netzwerke".

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 197–226, 2009.

We show that in a special case the construction gives sparse fault-tolerant connectors with respect to edge/node failures. We also present some existence results.

In Section 3 we study problems of **Fault Diagnosis in Multiprocessor Networks**. In [44] Preparata *et al.* introduced a graph theoretical model for *system-level diagnosis* (see a survey [30]), in which processors perform tests on one another via links in the system. Fault-free processors correctly identify the status of tested processors, while the faulty processors can give arbitrary test results. The goal is to identify faulty processors based on the test results. A system is said to be t-diagnosable if faulty units can be identified, provided the number of faulty units present does not exceed t.

We described an efficient *Diagnosis Algorithm* (DA) for fault identification in large interconnection networks. The algorithm has best known performance: it is linear in time and can be used for sequential diagnosis strategy, as well as for incomplete diagnosis in one step. The algorithm applied to arbitrary topology based interconnection systems G with N processors has sequential diagnosability $t_{\mathbf{DA}}(G) \geq \lceil 2N^{\frac{1}{2}} \rceil - 3$, which is optimal in the worst case.

We show that the problems related to diagnosability of a system have deep relationship to several challenging combinatorial optimization problems.

Section 4 is devoted to **Unconventional Error-Correcting Codes** in communication systems. When using amplitude modulation for error-correcting block codes, in several communication systems the magnitude of an error signal is small while the range of error signals can be large. In this case it is impractical to use known classical error-correcting codes. This is a motivation for the development of *codes correcting errors of a limited magnitude* introduced by Ahlswede *et al.* [5].

We study q-ary codes correcting all unidirectional errors (UEC-codes) of a given magnitude. Tight upper and lower bounds for the cardinality of those codes are obtained and their asymptotic growth rate is determined. For arbitrary code length and alphabet size q, near optimal constructions for UEC-codes capable of correcting all errors of a given magnitude are obtained. An infinite class of optimal codes, for arbitrary code length, is constructed. Recently these codes have been shown to be applicable for design of reliable Multilevel Flash memories. Several physical effects that limit the reliability and performance of Multilevel Flash memories induce errors that have low magnitude and are dominantly unidirectional.

In Section 5 we study **Parallel Error-Control Codes**. In [11] Ahlswede *et al.* introduced a new code concept for multiple-access channels (MAC) with a special error control mechanism. A communication channel consists of several sub-channels transmitting simultaneously and synchronously. The senders encode their messages into codewords of the same length over the same alphabet and transmit them in parallel. When an error occurs in a line at time T, then with a relatively high probability, an error also occurs in its neighbor lines. A parallel t-error-correcting code is a code capable of correcting all t or less errors of this type. Our main results here are constructions of optimal and near optimal parallel codes with simple decoding schemes. Nontrivial bounds and efficient constructions of such codes for two type of Z–channels have been obtained.

2 Connectors in Communication Networks

An (n, N)– communication network is defined here as a directed acyclic graph with n distinguished vertices called *inputs* and N other distinguished vertices called *outputs.* All other vertices are called *links.* A *route* in a network is a directed path from an input to an output. The *size* of a network is the number of edges, and the *depth* is the length of the longest route in it. An (n, N, d)*–connector*, also called a *rearrangeable network*, is a network of depth d $(n \leq N)$, such that for every injective mapping of the set of input vertices into a set of output vertices there exist n vertex disjoint paths joining each input to its corresponding output. Usually the size, in some approximate sense, corresponds to the cost and the depth corresponds to the delay of a communication network. Therefore, for the networks intended for a certain communication task it is preferable to have small size and small depth.

Symmetric connectors, i.e. connectors with $n = N$, are well studied. Shannon [47] showed that the size of an (n, n, ∞)–connector (unlimited depth) is lower bounded by $\Omega(n \log n)$. Later Beneš [18] gave constructions of $(n, n, \log n)$–connectors of size $O(n \log n)$. In applications it is important to have connectors of a limited depth. Pippenger and Yao [42] obtained lower and upper bounds for the size of an (n, n, d)–connector: $\Omega(n^{1+1/d})$ and $\mathrm{O}(n^{1+1/d}(\log n)^{1/d})$, respectively. The best known explicit construction for odd depth $2i+1$ has size $\mathrm{O}(n^{1+1/(i+1)})$ and is due to Pippenger [41]. Hwang and Richards [31] and Feldman, Friedman, and Pippenger [26] gave explicit constructions for depth 2 connectors of size $\mathrm{O}(n^{5/3})$. The latter can be used for construction of connectors of depth $2i$ and size $\mathrm{O}(n^{1+2/(3i-1)})$. For asymmetric connectors Oruc [40] gave constructions for depth $\Omega(\log_2 N + \log_2^2 n)$ of size $\mathrm{O}(N + n \log_2 n)$.

Explicit constructions for $(n, N, 2)$–connectors of size $(1 + o(1))N\sqrt{n}$ for $n \leq \sqrt{N}$ (and N= square of a prime) are given in [31] (see also [43]).

Asymmetric connectors of depth two are of particular interest in the design of sparse electronic switches. They are also useful as building blocks in multistage connectors.

Baltz, Jäger, and Srivastav [16] gave construction of $(1 + o(1))N\sqrt{3n/4}$ size connectors for all $n \leq \sqrt{N}$, and have shown, by a probabilistic argument, the existence of $(n, N, 2)$–connectors of size $\mathrm{O}(N)$, if $n \leq N^{1/2-\varepsilon}$, $\varepsilon > 0$.

A challenging problem is to construct linear–sized $(n, N, 2)$–connectors (even for some restricted values of n and N).

2.1 Existence Results

The existence of linear sized $(n, N, 2)$–connectors for $n \leq N^{1/2-\varepsilon}$, with $\varepsilon > 0$ is shown in [16]. The probabilistic method gives also tight upper bound for the size of an $(n, N, 2)$-connector.

Theorem 1. *[2] For all $n \geq 2$ and $N \geq N_0(n)$ there exist $(n, N, 2)$–connectors of size $2N(1 + o(1))$ and this is asymptotically optimal.*

It is natural to ask for the size of $(n, N, 2)$–connectors with $N^{1/2} \le n < N$. Construction of such connectors of size $O(N^{3/4}n)$ are given in [31], [16].

Theorem 2. *[2] For the size of an* $(N^\alpha, N, 2)$*–connector with* $1/2 \le \alpha < 1$ *we have lower and upper bounds:* $\Omega(N^{\alpha+1/2})$ *and* $O(N^{\alpha+1/2}\log N)$ *respectively.*

The proof follows from the following simple observation. Let G_1 and G_2 be $(n_1, N, 2)$ and $(n_2, N, 2)$–connectors respectively and let $G_1 * G_2$ be the $(n_1 + n_2, N, 2)$–network obtained by identifying the outputs of G_1 and G_2 by any one-to-one mapping. It is easy to see that $G_1 * G_2$ is an $(n_1 + n_2, N, 2)$–connector and the size of the resulting connector equals to the sum of sizes of G_1 and G_2. Suppose there exists an $(N^\alpha, N, 2)$–connector G of size $\Omega(N^x)$ with $1/2 \le \alpha \le 1$. We construct an $(N, N, 2)$–connector from $G * \ldots * G$, taking sufficiently many copies of G and then deleting all but N input vertices of the resulting network. The constructed connector has size $\Omega(N^{1-\alpha}N^x)$. This, together with the lower bound $\Omega(N^{3/2})$ in [26] for an $(N, N, 2)$-connector, implies that G has size $\Omega(N^{1/2+\alpha})$. Similarly the existence of linear-sized $(N^\delta, N, 2)$–connectors for any $0 < \delta < 1/2$ implies also the existence of $(N^\alpha, N, 2)$– connectors of size $\mathrm{O}(N^{1+\alpha-\delta})$ with $1/2 \le \alpha \le 1$. This can be used to obtain upper bounds for the size of $(N^\alpha, N, 2)$ in particular, connectors of size $\mathrm{O}(N^{\alpha+1/2}\log N)$.

2.2 Construction of Asymmetric Connectors

We describe a simple combinatorial construction of sparse asymmetric connectors. But first we need some preliminaries. For integers $a < b$ we denote $[a, b] = \{a, a{+}1, \ldots, b\}$ and $[0, \infty] := \{0\} \cup \mathbb{N}$. We denote $S(k, q) := \{(x_1, \ldots, x_k) : x_i \in [0, q]\}$ and for $q = \infty$ we use the notation $S(k, \infty)$. We define now a partial ordering on elements of $S(k, q)$ as follows. For $x, y \in S(k, q)$ we say that $x \le y$ if either $x_i = y_i$ or $x_i = 0$ for all $i = 1, \ldots, k$. Define also $r(x) =$ the number of nonzero coordinates of $x \in S(k, q)$ (note that $r(x)$ is usually called the Hamming weight of x). Thus $S(k, q)$ is a partially ordered set ordered by $\le$ with the rank function $r(x)$ defined for each element $x \in S(k, q)$. In the literature $S(k, q)$ is usually called the product of stars (see e.g. [25]). By $S_r(k, q)$ we denote the elements of rank r, that is $S_r(k, q) = \{x \in S(k, q) :\ r(x) = r\}$. Thus $S(k, q) = S_0(k, q) \dot\cup S_1(k, q) \dot\cup \cdots \dot\cup S_k(k, q)$, where $|S_i(k, q)| = \binom{k}{i} q^i,\ i = 0, 1,\ \ldots, k$.

Given integers $1 \le l < r \le k$ and q (or $q = \infty$), the l-th shadow of $x \in S_r(k, q)$ is defined by $\partial_l x = \{y \in S_{r-l} :\ x \ge y\}$. Correspondingly for $X \subset S_r(k, q)$, $\partial_l X = \{\partial_l x :\ x \in X\}$.

Next we define a linear order on $S(k, q)$. Define first $x(t) = \{i \in [k] : x_i = t\},\ x \in S(k, q)$. Recall also the colexicographic order on the subsets of $[1, k]$. For $A, B \subset [1, k]$ we say $A \prec_{col} B$ iff $\max((A \setminus B) \cup (B \setminus A)) \in B$. Now for $x, y \in S(k, q)$ we define the linear ordering $\prec_L$ as follows: $x \prec_L y$ iff $x(t) \prec_{col} y(t)$, where t is the greatest number such that $x(t) \ne y(t)$.

For a subset $X \subset S(k, q)$ let $\mathcal{C}(m, X)$ denote the set of the first m elements of X with respect to the ordering $\prec_L$. In our construction we use the following

Lemma 1. *For integers $1 \le l < r \le k$, q and a subset $A \subset S_r(k,\infty)$ with $|A| \le q^r\binom{k}{r}$ we have*

$$|\partial_l A| \ge \frac{|A|\binom{r}{l}}{\binom{k-r+l}{l}q^l}. \tag{2.1}$$

In particular, for $A \subset S_k(k,\infty)$ with $|A| \le \lfloor k/l \rfloor^k$ we have $|\partial_l A| \ge |A|$.

The lemma is a consequence of the following result due to Leeb.

Theorem L. [37] *For integers $1 \le r \le k$, m and a subset $A \subset S_r(k,\infty)$ with $|A| = m$ holds*

$$\partial_l \mathcal{C}(m, S_r(k,\infty)) \subseteq \mathcal{C}(|\partial_l A|, S_{r-1}(k,\infty)). \tag{2.2}$$

One of approaches for construction of connectors is the concatenation of a connector with a *concentrator*. An (N, L, c)–concentrator is an (N, L)–network such that for every set of $j \le c$ inputs there exist j disjoint routes containing these inputs. For concentrators of depth one (that is for bipartite graphs) this is equivalent to the property that every $j \le c$ input vertices have at least j neighbors, that is Hall's matching condition is satisfied for every set of $j \le c$ input vertices.

Depth-one concentrators, also called *crossbar concentrators*, are useful devices in designing of communication networks.

We are prepared now to describe our construction. Let the vertex set $V = \mathcal{I} \cup \mathcal{L} \cup \mathcal{O}$ of a graph $G = (V, E)$ be partitioned into input vertices $\mathcal{I}$ with $|\mathcal{I}| = n$, link vertices $\mathcal{L}$ with $|\mathcal{L}| = L$ and output vertices $\mathcal{O}$ with $|\mathcal{O}| = N$. Consider a network satisfying the following two conditions.

C1: $\mathcal{I}$ and $\mathcal{L}$ form a depth one connector (which clearly is a complete bipartite graph).

C2: $\mathcal{O}$ and $\mathcal{L}$ form an (N, L, n)-concentrator.

It is easy to see that G is an $(n, N, 2)$–connector.

Given $t > 2$ and $n \ge t^t$, let k be the minimum integer such that $n \le t^k$. Suppose $k = tl + r$ where $0 \le r < t$. Thus $t^{k-1} < n \le t^k$. Let also, for ease of calculations, $N = q^k$ (in general, $N = \Omega(q^k)$) for some integer $q > \binom{k}{l}^{1/l}$. We construct the following network satisfying conditions C1, C2:
$\mathcal{O} := \mathcal{C}(N, S_k(k,q))$, $\mathcal{L} := \partial_l \mathcal{C}(N, S_k(k,q))$, $L := |\mathcal{L}|$, and $n := |\mathcal{I}| = \Theta(t^k)$. The edge set E is defined in a natural way: for $x \in \mathcal{O}$ and $y \in \mathcal{L}$ we have an edge $(x,y) \in E$ iff $y \in \partial_l(x)$.

In view of Lemma 1, for any subset $X \subset \mathcal{O}$ with $|X| \le n$ we have $|\Gamma(X)| \ge |X|$. Hence $\mathcal{O}$ and $\mathcal{L}$ form an (N, L, n)-concentrator. The size of the connector

$$|E| = Ln + N\binom{k}{l} = q^{k-l}\binom{k}{l}\Theta(t^k) + q^k\binom{k}{l}.$$

We choose any $q \ge t^t$. Easy calculations show that we have a $(\Theta(t^k), t^{tk})$–connector with

$$|E| \le 2N\binom{k}{l} = 2Nn^{\frac{1}{t}(1+o(1))}.$$

In particular, for $l = 1$, $n \leq k^k \leq q$ we get $|E| \leq 2N \log n/(1 + o(1)) \log\log n$. Thus we have

Theorem 3. *[2] For all integers $t > 2$, $n \geq t^t$, and $N = \Omega(n^t)$ the construction above gives $(n, N, 2)$-connectors of size $Nn^{\frac{1}{t}(1+o(1))}$. In particular, for all n and $N > N(n)$ we get connectors of size $2N \log n/(1 + o(1)) \log\log n$.*

An important task in designing of communication networks is reliability. Therefore it is natural to consider connectors which are robust under edge (node) failures. An $(n, N, 2)$–connector is called t–edge fault–tolerant if in spite of deletion of any t or less edges (edge failures) the resulting graph is an $(n, N, 2)$–connector. Correspondingly, it is called t– fault–tolerant if this property holds after deletion of any t or less link vertices (node failures), which also implies t–edge fault–tolerance.

Note that for any t–edge fault–tolerant connector, t is less than its minimum degree of input/output vertices.

Construction of fault-tolerant $(n, N, 2)$–connectors of size $(1 + o(1))N \log_2 n$ is similar to the construction described above. Here we use the Boolean lattice of finite subsets of $[1, N]$, $N \subset \mathbb{N}$, and output vertices are associated with the k–sets of $[1, N]$.

Theorem 4. *[1] For all N and $n = O\big(N^{1/\sqrt{\log_2 N}}\big)$ we have explicit construction of $(n, N, 2)$–connectors of size $(1 + o(1))N \log_2 n$ which are $(k - 1)$– fault–tolerant, where $k = \Theta(\log_2 n)$ is the degree of output vertices.*

2.3 Open Problems

A challenging open problem is an explicit construction of linear-sized depth two (or at least limited depth) asymmetric connectors.

In our construction we used posets of star products. Can we improve the construction using other posets? In fact, our approach above reduces to construction of sparse concentrators of depth one. In general, we have the following combinatorial optimization problem which has been extensively studied in various settings (see [30],[54], [41]). Given N, l, c determine

$E(N, l, c)$:= the minimum possible size of a depth-one (N, l, c) –concentrator.

This, however seems to be a difficult problem.

Note that in terms of the adjacency matrix A of a bipartite graph $G = (V, E)$ $E(N, l, c)$ is the minimum number of 1's of an $l \times N$ (0,1)–matrix A such that the boolean sum of every j columns, $j = 1, ..., c$, has at least j ones. Equivalently, no set of j columns, $j = 1, ..., c$, contains an $(l - j + 1) \times j$ all zero submatrix.

A weakened version of the problem is as follows: Minimize the number of 1's of an $l \times N$ (0,1)–matrix A such that it does not contain an $(l - n + 1) \times n$ all zero submatrix.

This problem (exchanging $0 \leftrightarrows 1$) is equivalent to an old problem (see [20]) called *Zarankiewicz's problem* (1951): Given integers k, m, a, b; $0 \leq a \leq k$, $0 \leq b \leq m$, determine

$Z_{a,b}(k, m)$:= maximum number of 1's in an $k \times m$ (0,1)– matrix which does not contain an $a \times b$ all one submatrix.

The problem is widely open, even for the case $a = b = 2$. Kövari, Sós, and Turán [36] obtained the following upper bound: $Z_{a,b}(k, m) \leq (a-1)^{1/b}(m - b + 1)k^{1-1/b} + (b-1)k$.

3 Fault Diagnosis in Large Multiprocessor Networks

With the continuing development of semiconductor technologies, large multiprocessor systems such as VLSI have been of growing concern. A multiprocessor system may contain a huge number of processors proceeding simultaneously at very high speed. The uninterrupted processing is an important task in designing of reliable multiprocessors systems. An integral part of reliability is the identification of faulty processors.

The concept of *system–level* diagnosis was introduced by Preparata, Metze, and Chien [44] to perform automatic fault diagnosis in multiprocessor systems. In their graph theoretical model, called PMC model, a system S is composed of independent units $u_1, \ldots, u_n$ connected by communication links. The system is represented as an undirected graph $G = (V, E)$, where the vertices are units and edges are interconnection links. In the PMC model diagnosis is based on a suitable set of tests between units. A unit u_i can test u_j iff the vertices corresponding to u_i and u_j in the graph $G = (V, E)$ of the system S are adjacent. The outcome of a test in which u_i tests u_j is denoted by a_{ij}, where $a_{ij} = 1$ if u_i finds u_j to be faulty and $a_{ij} = 0$ if u_i finds u_j to be fault-free.

The basic conditions of the PMC model are the following:

- The fault-free units give correct test outcomes.
- The answers of faulty units are unreliable.
- The number of faulty units t is bounded.

The set of tests for the purpose of diagnosis is represented by a set of directed edges where the presence of oriented edge (u_i, u_j) means that u_i tests u_j. Given a faulty set of units $F \subset V$ the set of all test outcomes $\{a_{ij}\}$ is called *syndrome.* The task is to identify the faulty units based on a syndrome produced by the system. In [44] two different kinds of strategies were introduced for implementing the diagnosis approach.

One-step diagnosis (or diagnosis without repair): a system is called t-fault diagnosable (or shortly t-diagnosable) in one step, if all faulty units can be uniquely identified from any syndrome, provided the number of faulty units does not exceed t.

Sequential diagnosis (or diagnosis with repair): a system is called sequentially t-diagnosable if it can identify at least one faulty unit from any syndrome, provided the number of faulty units does not exceed t. Under a sequential diagnosis strategy a system can locate a faulty unit, repair it and then repeat the process until all faulty units are repaired.

The *degree of diagnosability*, or simply diagnosability, of a system graph G is defined (for both kinds of strategies) as the maximum t such that the system is t-diagnosable.

The PMC model has been widely studied (see [17] for a good survey). It is known that the maximum degree of diagnosability of a one-step diagnosis algorithm for any system is bounded from above by the minimum vertex degree of the interconnection graph. However, the real commercial multiprocessor systems are based on topologies of graphs with small average vertex degree (like grids, hypercubes, cube-connected cycles, trees etc).

Sequential diagnosis is a much more powerful strategy than one-step t-fault diagnosis. On the other hand the sequential diagnosis has the disadvantage of repeated execution of diagnosis and repair phases and may be time consuming for large systems.

That was the motivation for developing diagnosis algorithms (see [21]) which are able to diagnose in one step the status of a large fraction of the system units (i.e. if a "large" subset F' of the actual fault set F can be identified from any syndrome, provided $|F| \leq t$). This approach is referred to as *incomplete diagnosis in one step.*

In fact there are two main problems (for both strategies). Theoretical: determination of diagnosability of a given system and Algorithmic: development of algorithms for fault identification.

Note that the problem of determining the sequential diagnosability of a system is shown to be co-NP complete [49].

The *diagnostic graph DG* of a system graph $G = (V, E)$, corresponding to a given syndrome, consists of bidirectional arcs, between every two neighbors of the original graph G, labelled by 0 or 1. Let $\{u, v\} \in E(G)$, then the presence of oriented edges (u, v) and (v, u) with $a_{uv} = 1$ and $a_{vu} = 0$ implies that v is faulty. Thus, in the worst case analysis, we assume that the outcomes of any two neighbors coincide. Therefore, a diagnostic graph is represented as anssh xde16 -X undirected graph where each edge is labelled by a 0 or a 1.

Given a syndrome, a subset F of the vertex set V is called a *consistent fault set* if the assumption that the vertices in F are faulty and those in $V \setminus F$ are fault–free is consistent with the syndrome. The following simple facts are useful for obtaining upper and lower bounds for the diagnosability of a system graph.

Proposition 1. *Given a syndrome, let $F_1, \ldots, F_k$ be a collection of consistent fault sets with $|F_i| \leq t$ for $i = 1, \ldots, k$. Then G is not sequentially t-diagnosable if $\bigcap_{i=1}^{k} F_i = \varnothing$ and $\bigcup_{i=1}^{k} F_i = V$.*

Given a diagnostic graph DG, define the subgraph G_0 consisting of edges labelled only by 0 (0-edges). The connected components of the graph G_0 are called *0-components* of DG.

Proposition 2. *(i) All vertices of a 0-component in a diagnostic graph DG have the same status: "faulty" or "fault-free".*
(ii) Suppose the size of a largest 0-component $K \subset$ in G_0 is greater than the fault bound t. Then all vertices of K can be identified as fault-free.

3.1 Two Extremal Problems on Graphs

Motivated by a problem (Dijkstra's critical section problem) arising in parallel computation for unreliable networks, Ahlswede and Koschnick [15] considered the following extremal problems for graphs.

Problem 1. *Given a connected graph $G = (V, E)$, let $\lambda(G, c)$ denote the maximal number such that removal of any $\lambda(G, c)$ or less vertices results in a graph with a connected component of size at least c. Determine or estimate $\lambda(G, c)$.*

Problem 2. *Removing edges instead of vertices, define analogously the function $\mu(G, c)$ and determine or estimate $\mu(G, c)$.*

Define also the function $\lambda^*(G, c)$ (resp. $\mu^*(G, c)$) = minimal number with the property that there exist $\lambda^*(G, c)$ vertices (resp. $\mu^*(G, c)$ edges) whose removal results in a graph with a maximal connected component of size $\leq c$. Observe that $\lambda^*(G, c)$=$\lambda(G, c + 1) + 1$ and $\mu^*(G, c)$= $\mu(G, c + 1) + 1$.

In fact, these functions are measures of connectivity in a graph, which generalize the known notion of edge/vertex connectivity in graphs.

It is not hard to show that both problems are NP-hard.

We note that both functions $\lambda(G, c)$ and $\mu(G, c)$ are useful for diagnosis problems in multiprocessor systems. In fact the following derived quantity is essential. For a graph G define $m(G) = \max\{x : \lambda(G, x + 1) \geq x\}$. Proposition 2 implies now

Proposition 3. *For every interconnection graph G, the diagnosability $t(G) \geq m(G)$.*

Note, however, that in general $m(G)$ can be much smaller than the degree of sequential diagnosability. Consider, for example, a star graph G on $N = 2k + 1$ vertices. It is not hard to observe that the sequential diagnosability of this graph $t(G) = k$ while $m(G) = 0$.

Khanna and Fuchs [33], and also Caruso *et al.* [21], studied the function $m(G)$ and gave algorithms for fault identification for some regular structures. In [33] a sequential diagnosis algorithm (referred to as PARTITION) applied to arbitrary interconnection graph on N vertices has diagnosability $\Omega(N^{\frac{1}{3}})$. Yamada *et al.* [55] described a sequential diagnosis algorithm (referred to as HYBRID) for an interconnection graph G on N vertices with diagnosability $t_{HYBRID}(G) \geq \lceil \sqrt{N-1}\, \rceil - 1$.

Next we describe an efficient diagnosis algorithm DA [4] which can be used for sequential diagnosis as well as for incomplete diagnosis in one step. In particular, the algorithm applied to arbitrary topology based interconnection systems has the best performance.

3.2 Diagnosis Algorithm DA

Given a connected graph $G = (V, E)$ and a syndrome, that is, a diagnostic graph $DG = (V, E')$, where each edge of E is labelled by a 0 or 1.

Step 1. Partition the vertices of DG into 0-components $K_1, \dots, K_\ell$; $\mathcal{K} := \{K_1, \dots, K_\ell\}$.
Step 2. Construct the contracted graph $G_c = (V_c, E_c)$ as follows.

Each component K_i contracts to vertex $a_i \in V_c$ and $\{a_i, a_j\} \in E_c$ iff there is an edge $\{u, v\}$ (labelled with 1) in E with $u \in K_i$ and $v \in K_j$. To each vertex a_i of V_c assign the weight $wt(a_i) = |K_i|$. Thus G_c is an undirected graph with weights on vertices. Clearly $\sum\limits_{a \in V_c} wt(a) = |V|$. The weight of a subgraph $G' \subset G_c$ is defined by $wt(G') = \sum\limits_{b \in V'} wt(b)$, where V' is the vertex set of G'.
Step 3. Find a spanning tree TG_c of G_c.
Step 4. Partition the vertex set of TG_c into subsets $T_1, \dots, T_p$, each containing at least two vertices, such that the induced subgraph of each subset T_i forms a star S_i, $i = 1, \dots, p$. Denote by z_i the center of S_i, $i = 1, \dots, p$ and put

$$w_i := \min\bigl\{wt(z_i), wt(S_i \setminus \{z_i\})\bigr\}, \alpha_i := \max\bigl\{wt(z_i), wt(S_i \setminus \{z_i\})\bigr\}, i=1, \dots, p,$$
$$\bar{w} := w_1 + \cdots + w_p, \quad \bar{\alpha} := \alpha_1 + \cdots + \alpha_p.$$

Step 5. Determine $\Delta = \max_{1 \le i \le p}\{\alpha_i + \bar{w} - w_i\}$. Suppose $\Delta = \alpha_r + \bar{w} - w_r$; $r \in [1, p]$. Suppose also the number of actual faults $t \le \Delta - 1$.
Step 6. If $wt(z_r) = w_r$, then the vertex z_r is labelled as "faulty". The component $K_{i_r} \subset \mathcal{K}$ corresponding to z_r is diagnosed as faulty set.

If $wt(z_r) = \alpha_r$, then z_r is labelled as "non-faulty" and the remaining vertices of S_r are labelled as "faulty". The components corresponding to vertices $S_r \setminus \{z_r\}$ are diagnosed as faulty sets.

The described algorithm allows to identify the status of at least one vertex if the number of faulty units $t < \min \Delta(t, G)$, where the minimum is taken over all syndromes produced by all consistent faulty sets $F \subset V$ with $|F| \le t$.

The status of the remaining vertices is identified iteratively applying the "diagnosis and repair" procedure.

Theorem 5. *[4] Given interconnection graph $G = (V, E)$,*
(i) the overall running time of the diagnosis algorithm DA is $O(|E|)$,
(ii) it requires at most $d(G)$ (diameter of G) iterations, to identify all faults,
(iii) and given a lower bound $m^*(G)$ for $m(G)$, the diagnosability of the algorithm
$t_{DA}(G) \ge \max\{m^*(G), 2|V|^{\frac{1}{2}} - 3\}$.

Corollary 1. *For an arbitrary interconnection graph G on N vertices the diagnosability of the algorithm $t_{DA}(G) \ge \lceil 2N^{\frac{1}{2}} \rceil - 3$.*

In fact, the algorithm is optimal for "bad graphs": there exist infinitely many interconnection graphs $G = (V, E)$ with sequential diagnosability $\lceil 2|V|^{\frac{1}{2}} \rceil - 3$.

In particular there are such k–trees.

Example. Let $k = N^{\frac{1}{2}}$ be an integer and let DG be a diagnostic graph on N vertices shown in Figure 1.

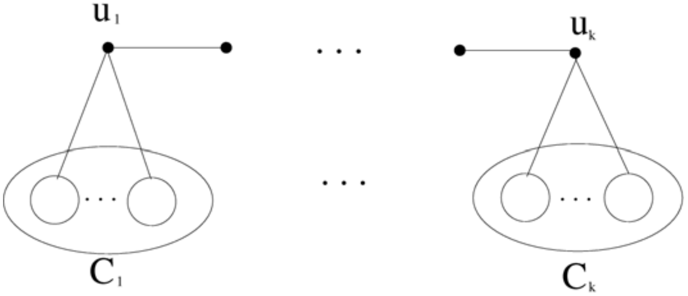


Fig. 1.

DG: each set of vertices C_i with $|C_i| = k-1$, $(i = 1, \ldots, k)$ represents a union of some 0–components (denoted by circles), where the edges incident with vertices $u_1, \ldots, u_k$ are labelled by 1's. We denote $U = \{u_1, \ldots, u_k\}$. and define then the faulty sets $F_1, \ldots, F_k$ as $F_i = (U \backslash \{u_i\}) \cup C_i,\ i = 1, \ldots, k$. Note that $|F_i| = 2k-2$. All these sets are consistent fault sets (their intersection is empty and the union is the vertex set of G). Therefore, G is not sequentially $(2N^{\frac{1}{2}} - 2)$–diagnosable.

3.3 Bounds for $\lambda(\mathcal{H}_n, c)$ in Hamming Graphs $\mathcal{H}_n$

Lower bound. Let $\mathcal{H}_n = \{0,1\}^n$ denote the binary Hamming space and let $d(x,y)$ denote the Hamming distance between any two vectors $x, y \in \mathcal{H}_n$, defined as the number of coordinates in which they differ. We associate $\mathcal{H}_n$ with the Hamming graph $G(\mathcal{H}_n)$ where two vertices $x, y \in \mathcal{H}_n$ are adjacent iff $d(x,y) = 1$. Let us denote $N_{n,k+1} = \binom{n}{n} + \cdots + \binom{n}{k+1}$,

Theorem 6. *(i) For $n \geq 2k$ we have*

$$\lambda^*(n, N_{n,k+1}) = \binom{n}{k}$$

(ii)

$$\lambda(n, N_{n,k+1}) = \begin{cases} \binom{n}{k} & ,if \quad n > 2k \\ \binom{n}{k} + 1 & ,if \quad n = 2k,\ \ k \geq 3. \end{cases}$$

The proof is based on Harpers vertex isoperimetric theorem [29].

Upper bound. We describe a regular separation of the vertices of the Hamming graph $G(\mathcal{H}_n)$. For convenience of the description, we identify $\mathcal{H}_n$ with the set of vectors $\mathcal{H}_n^* := \{-1,1\}^n \subset \mathbb{R}^n$ using $1 \to -1$ and $0 \to 1$ exchange of the coordinates. Thus we can speak about an identical graph $G(\mathcal{H}_n^*)$. Note that the Hamming distance between any $x, y \in \mathcal{H}_n^*$ can be evaluated by their inner product $\langle x, y \rangle$, that is, $d(x,y) = \frac{1}{2}(n - \langle x, y \rangle)$.

The idea is to separate the elements of $\mathcal{H}_n^*$ into equal sized parts by mutually orthogonal hyperplanes of $\mathbb{R}^n$. It is known that for any $n = 2^k$ there exist Hadamard matrices of order n. Recall that a $(+1,-1)$–matrix H of size $n \times n$ is called a Hadamard matrix of order n, if $HH^T = nI_n$. Hadamard matrices H_n of order $n = 2^k$ can be constructed as k-th Kronecker power of matrix $H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. Note that the corresponding (0,1)-matrix without all-zero column can be viewed as the simplex code of length $2^k - 1$ (well known in Coding Theory [39]) with a generator matrix of size $k \times 2^k - 1$ consisting of all nonzero column vectors.

Given a set of n vectors $v_1, \dots, v_n \in \mathcal{H}_n^*$, let $\langle v_1 \rangle, \dots, \langle v_n \rangle$ be the hyperplanes defined by $\langle v_i \rangle = \{x \in \mathbb{R}^n : \langle v_i, x \rangle = 0\}$, $i = 1, \dots, n$.

Given an integer $1 \leq r \leq n$ let us define the set of *sign sequences* $\Sigma := \{+,-\}^r$. Let $x \in \mathcal{H}_n^*$ and let $(\sigma_1, \dots, \sigma_r) \in \Sigma$. We say that $Sign(x) = (\sigma_1, \dots, \sigma_r)$ if $Sign\langle x, v_i \rangle = \sigma_i$, $i = 1, \dots, r$, (where for a real number a, like $\langle x, v_i \rangle$, $Sign\ a$ is defined in the natural way). Let $\Sigma_1, \dots, \Sigma_{2^r}$ be the elements of Σ in some fixed order. Define the sets $B_i = \{x \in \mathcal{H}_n^* : Sign(x) = \Sigma_i\}$; $i = 1, \dots, 2^r$. Clearly these sets are disjoint. Denote the set of remaining elements of $\mathcal{H}_n^*$ by S_r, that is, $S_r = \{x \in \langle v_i \rangle \cap \mathcal{H}_n^*\colon 1 \leq i \leq r\}$. The hyperplanes $\langle v_1 \rangle, \dots, \langle v_r \rangle$ separate the points of $\mathbb{R}^n$ into classes which have different signs. Therefore we have the following.

Lemma 2. *S_r is a vertex separating set for $B_1, \dots, B_{2^r}$, that is, any path between the vertices of two distinct classes B_i and B_j contains a vertex of S_r.*

Theorem 7. *Given integers $n = 2^k$ and $1 \leq r \leq k$, we have*

$$\lambda(n, 2^{n-r} - |S_r|/2^r) \leq |S_r|. \tag{3.1}$$

Corollary 2. *For positive integers n and $r \leq \lfloor \log n \rfloor$.*

$$\lambda(n, 2^{n-r}) = O(r2^n/\sqrt{n}). \tag{3.2}$$

Conjecture. *For $n = 2^k$ and $1 \leq r \leq k$*

$$\lambda(n, 2^{n-r} - |S_r|/2^r) = |S_r|. \tag{3.3}$$

Note that (in view of Theorem 6) the conjecture holds for $r = 1$.

3.4 Diagnosability of the n-Cube

Theorem 6 has several consequences. Suppose the number of faulty sets $\binom{n}{k-1} < t \leq \binom{n}{k}$, $(k \leq n/2)$, then there exists a set of vertices $A \subset \mathcal{H}_n$ with $|A| \geq N_{n,k+1}$ that can be identified as "fault-free" and the vertices ΓA can be identified as "faulty". Thus the status of at least $|\sigma(A)|$=$|A \cup \Gamma A|$ elements can be identified in one step.

Corollary 3. *(i) Let t be the number of faulty vertices and let $\binom{n}{k-1} < t \le \binom{n}{k}$, $k \le n/2$. Then the status of at least $N_{n,k}$ vertices can be identified in one step. In particular, for $k = n/2$, the status of at least $N_{n,n/2} = 2^{n-1} + \binom{n-1}{\frac{n}{2}-1}$ vertices can be identified.*

(ii) Given integer $n \ge 3$ we have $m(\mathcal{H}_n) \ge \binom{n}{\lfloor \frac{n}{2} \rfloor}$ and hence the degree of sequential diagnosability of the n-cube $t(\mathcal{H}_n) > \binom{n}{\lfloor \frac{n}{2} \rfloor}$

An important parameter in sequential diagnosis is the number of test and repair iterations needed to locate all the faulty units within the system (see [21], [46]). Thus, reducing the number of iterations is an important task in implementation of a diagnosis scheme. It was shown in [46] that this number for n-cubes is upper bounded by $\Theta(n)$. As a direct consequence of Theorem 6 we get

Corollary 4. *Let $\binom{n}{k-1} < t \le \binom{n}{k}$, $k \le n/2$, then the number of iterations needed for sequential diagnosis is at most k.*

Theorem 7, in turn, can be used to obtain an upper bound for the sequential diagnosability of the n-cube. The following upper bound by Yamada et al. [55] can be easily derived from (2.13).

Theorem 8. *[55] $t(\mathcal{H}_n) = O(2^n \log n/\sqrt{n})$.*

3.5 Diagnosis under Pessimistic Strategy: $t|s$-Diagnosis

In both, one-step and sequential diagnosis strategies, it is assumed that only those processors that where truly faulty were replaced. Therefore, the strategy may be called *precise diagnosis strategy.*

Friedman [27] proposed a strategy under which up to s or less processors containing all (t or less faulty processors) and possibly some processors of unknown status were identified and replaced. This strategy is called *pessimistic diagnosis strategy* or shortly *$t|s$–diagnosis.* A system is called $t|s$–diagnosable if for a given syndrome all faulty units can be isolated within a set of at most s units, provided the number of faulty units does not exceed t.

The motivation for the study of such strategy is to increase the "diagnosability" of a given multiprocessor networks. Suppose all $t-1$ neighbors of a processor are faulty. Then under precise diagnosis strategy the status of this isolated processor cannot be determined. Under the pessimistic strategy such an isolated processor is treated as potentially faulty and replaced. Therefore the diagnosability under pessimistic strategy can be much higher.

Definition 1. *Given integer $r \ge 0$ the degree of $t|t+r$–diagnosability of a system is defined as the maximum t for which the system is $t|t+r$ diagnosiable.*

Kavianpour and Kim [32] showed that $t|t$– diagnosability of the n-cube is $2n-2$ for $n \ge 4$ (Note that the diagnosability of the n–cube under one-step strategy is n.). The next theorem gives exact answer for all $0 \le r \le 2n-4$.

Theorem 9. *The degree of $t|t+r$ diagnosability of the n–cube is*
$\binom{n}{2}-\binom{n-r-2}{2}+1$ *for* $0\leq r\leq n-2$; $n\geq 4$, *and is*
$\binom{n}{2}+\binom{n-2}{2}-\binom{2n-r-4}{2}+1$ *for* $n-1\leq r\leq 2n-4$; $n\geq 6$.

3.6 Open Problems

Close the gap between upper and lower bounds (or give better estimates) for the sequential diagnosability of n–cube systems.

A closely related problem is to give good estimates for $m(\mathcal{H}_n)$ and $\lambda(\mathcal{H}_n,c)$.

Consider these problems (and the $t|s$–diagnosis problem) for other popular topology based systems.

4 Unconventional Error–Correcting Codes

In the binary symmetric channel it is assumed that for both symbols of the alphabet the probability of an error is the same. However in many digital systems such as fiber optical communications and optical disks the ratio between probability of errors of type $1\rightarrow 0$ and $0\rightarrow 1$ can be large. Practically one can assume that only one type of errors, called asymmetric, can occur in those systems. This binary channel is referred to as Z–channel. Similarly, asymmetric errors are defined for a q-ary alphabet $Q=\{0,\dots,q-1\}$. For every input symbol i the receiver gets a symbol only from $\{i,\dots,q-1\}$. Thus for any transmitted vector $(x_1,\dots,x_n)$ the received vector is of the form $(x_1+e_1,\dots,x_n+e_n)$ where $e_i\in Q$ and $x_i+e_i\leq q-1$, $i=1,\dots,n$. For more information on asymmetric/unidirectional error correcting codes and their applications see [19], [35], [52].

When using amplitude modulation (in multilevel transmission) for error correcting block codes, in several communication systems the magnitude of an error signal (i.e. the correlation between an input and the output signal) is small while the range of error signals (i.e. the number of errors occurred in a block) can be large (even close to the block length). In this case it is impractical to use known classical error correcting codes. This is a motivation for the development of *codes correcting /detecting asymmetric errors of a limited magnitude.* These codes were first introduced and studied in Ahlswede *et al.* [5],[8].

Recently these codes have been shown to be applicable for design of reliable Multilevel Flash memories [22]. Several physical effects that limit the reliability and performance of Multilevel Flash memories induce errors that have low magnitude and are dominantly asymmetric. Flash Memory is a NonVolatile Memory (NVM) technology that is both electrically programmable and electrically erasable. This property, together with high storage densities and high speed programming, has made Flash Memory the dominant NVM technology and a prominent enabler for many portable applications and technologies. It is a technology that is primarily used in memory cards and USB flash drives, which are used for general storage and transfer of data between computers and other digital products.

We consider a special type of asymmetric errors in a q-ary channel, where the magnitude of each component of $\mathbf{e}$ satisfies $0 \leq e_i \leq \ell$ for $i = 1, \ldots, n$. We refer to ℓ as *level.* Correspondingly we say that *a unidirectional error of level* ℓ has occurred, if the output is either $\mathbf{x} + \mathbf{e}$ or $\mathbf{x} - \mathbf{e}$ (in the latter case, it is of course required that $x_i \geq e_i$ for all i).

If the error vector $\mathbf{e}$ has Hamming weight t, then we say that t errors of level ℓ have occurred. Thus the general coding problem can be formulated as follows.

Problem. Given n, ℓ, t, q construct q-ary codes of length n, capable of correcting t errors of level ℓ, and having maximum possible cardinality.

4.1 ℓ-AEC and ℓ-AUC Codes

We study *q-ary codes correcting all asymmetric errors of given level* ℓ (that is $t = n$), abbreviated as ℓ-AEC codes, and *codes correcting all unidirectional errors of level* ℓ, abbreviated as ℓ-UEC codes. We introduce first two distances in order to characterize the error correcting capabilities of ℓ-AEC/ℓ-UEC codes.

Definition 2. *For* $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in Q^n$ *and* $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in Q^n$*,*

$$d_{\max}(\mathbf{x}, \mathbf{y}) = \max\{|x_i - y_i| : i = 1, 2, \ldots, n\}$$
$$d_u(\mathbf{x}, \mathbf{y}) = \begin{cases} d_{\max}(\mathbf{x}, \mathbf{y}) & \text{if } \mathbf{x} \geq \mathbf{y} \text{ or } \mathbf{y} \geq \mathbf{x}, \\ 2d_{\max}(\mathbf{x}, \mathbf{y}) & \text{if } \mathbf{x} \text{ and } \mathbf{y} \text{ are incomparable,} \end{cases}$$

where $\mathbf{x} \geq \mathbf{y}$ *means that* $x_i \geq y_i$ *for all* i*.*

Proposition 4. *A code* $\mathcal{C} \subset Q^n$ *is an* ℓ*-AEC code iff* $d_{max}(\mathbf{x}, \mathbf{y}) \geq \ell + 1$ *for all distinct* $\mathbf{x}$,$\mathbf{y}$ *in* $\mathcal{C}$*.*

Proposition 5. *A code* $\mathcal{C} \subset Q^n$ *is an* ℓ*-UEC code if and only if* $d_u(\mathbf{x}, \mathbf{y}) \geq 2\ell + 1$ *for all distinct* $\mathbf{x}$, $\mathbf{y}$ *in* $\mathcal{C}$*.*

For given ℓ, let $A_a(n, \ell)_q$ and $A_u(n, \ell)_q$ denote the maximum number of words in a q-ary ℓ-AEC code, or ℓ-UEC code respectively, of length n.

Theorem 10. *For all integers* n *and each* $\ell \in Q$*, the code*

$$\mathcal{C} = \left\{(x_1, x_2, \ldots, x_n) \in Q^n : x_i \equiv 0 \mod (\ell + 1) \text{ for } i = 1, 2, \ldots, n\right\}$$

is an ℓ*-AEC code with* $|\mathcal{C}| = \left\lceil \frac{q}{\ell+1} \right\rceil^n$.

Note that a received vector can be decoded by component-wise rounding downwards to the nearest multiple of ℓ+1.

As any ℓ-UEC code is an ℓ-AEC code, Theorem 10 implies that

$$A_u(n, \ell)_q \leq A_a(n, \ell)_q = \left\lceil \frac{q}{\ell + 1} \right\rceil^n. \tag{4.1}$$

We give two constructions for q-ary ℓ-UEC codes valid for all pairs (q,ℓ). We denote by $Q_{\ell+1}$ all integers in $Q=[0,q-1]$ that are multiples of $\ell+1$, that is

$$Q_{\ell+1}=\{m\in\{0,1,\dots,q-1\}:m\equiv 0\ (\mathrm{mod}\ \ \ell+1)\}=\{a(\ell+1):0\le a\le b-1\}, \tag{4.2}$$

where $b=|Q_{\ell+1}|=\left\lceil\frac{q}{\ell+1}\right\rceil$.

It is clear that $d_{max}(\mathbf{x},\mathbf{y})\ge\ell+1$ for any two distinct words $\mathbf{x},\mathbf{y}$ in $Q_{\ell+1}^n$.

Construction 1. "Taking a subset of $Q_{\ell+1}^n$"
For each j let

$$C(j)=\{(x_1,x_2,\dots,x_n)\in Q_{\ell+1}^n:\sum_{i=1}^n\frac{x_i}{\ell+1}=j\}.$$

Any two distinct words from $C(j)$ clearly are incomparable and so $C(j)$ is an ℓ-UEC code. It is clear that

$$|C(j)|=|\{(y_1,y_2,\dots,y_n)\in\{0,1,\dots,b-1\}^n:\sum_{i=1}^n y_i=j\}|.$$

This construction leads to the following

Theorem 11. *For each integer q and $\ell\in Q$, there is a constant $c>0$ such that for each n,*

$$A_u(n,\ell)_q\ge c\frac{1}{\sqrt{n}}\lceil\frac{q}{\ell+1}\rceil^n\ .$$

Construction 2. "Adding tails to words from $Q_{\ell+1}^n$"

Proposition 6. *Let $X\subset Q^n$ be an ℓ-AEC code. For $\mathbf{x}\in X$, let $S(\mathbf{x})$ denote the sum of its entries, and let s_1,s_2 be such that for each $\mathbf{x}\in X$, $s_1\le S(x)\le s_2$. Let $\phi:[s_1,s_2]\to Q^m$ be such that for all $a,b\in[s_1,s_2]$ with $a>b$, there is an $i\in\{1,2,\dots,m\}$ such that $(\phi(a))_i<(\phi(b))_i$. Then $\mathcal{C}=\{(\mathbf{x},\phi(S(x)):\mathbf{x}\in X\}\subset Q^{n+m}$ is an ℓ-UEC code.*

Theorem 12. *For each q and ℓ, there exists a positive constant K such that for each n,*

$$A_u(n,\ell)_q\ge Kb^n n^{-\frac{1}{2}\log_q b},\ \ \textit{where } b=\lceil\frac{q}{\ell+1}\rceil\ .$$

In [51] Varshamov and Tennengolts gave the first construction of nonlinear codes correcting asymmetric errors. Given $n\in\mathbb{N}$ and an integer a, the Varshamov–Tennengolts code (VT code) $C(n,a)$ is defined by

$$C(n,a)=\big\{x^n\in\{0,1\}^n:\sum_{i=1}^n ix_i\equiv a\mod(n+1)\big\}. \tag{4.3}$$

Code $C(n,a)$ is capable of correcting all single asymmetric errors. Moreover it was shown that $|C(n,0)| \geq |C(n,a)|$ and

$$|C(n,0)| \geq \frac{2^n}{n+1}, \tag{4.4}$$

thus exceeding the Hamming upper bound for the size of a binary single symmetric error correcting code.

We study VT-type ℓ-UEC codes. Note, however, that unlike the VT-codes, the codes we introduce here are defined by means of some linear equation (rather than a congruence) over the real field. Namely given $Q = [0, q-1] \subset \mathbb{R}$ and $a_0, \dots, a_{n-1}, a \in \mathbb{Z}$ let

$$X = \{(x_0, \dots, x_{n-1}) \in Q^n : \sum_{i=0}^{n-1} a_i x_i = a\}. \tag{4.5}$$

Note that X defines an ℓ-UEC code iff for each distinct $\mathbf{x}, \mathbf{y} \in X$ holds $\mathbf{x} - \mathbf{y} \notin [-\ell, \ell]^n$ and $\mathbf{x} - \mathbf{y} \notin [0, 2\ell]^n$. Thus, a sufficient condition for the set of vectors $X \subset Q^n$ to be an ℓ-UEC code is that the hyperplane H defined by

$$H = \left\{ (x_0, \dots, x_{n-1}) \in \mathbb{R}^n : \sum_{i=0}^{n-1} a_i x_i = 0 \right\}$$

does not contain vectors from $[-\ell, \ell]^n \cup [0, 2\ell]^n$, except for the zero vector.

An ℓ-UEC code of VT type may have the advantage of a simple *Encoding and Decoding procedure*. In particular, let $\mathcal{C}$ be a code given by (3.5) where for $i = 0, 1, \dots, n-1$, $a_i = (\ell+1)^i$. Suppose for the received vector $\mathbf{y} = (y_0, \dots, y_{n-1})$ we have

$$\sum_{i=0}^{n-1} (\ell+1)^i y_i = a'$$

with $a' \geq a$. Then the transmitted vector $(x_0, \dots, x_{n-1}) = (y_0 - e_0, \dots, y_{n-1} - e_{n-1})$, where the error vector $(e_0, \dots, e_{n-1})$ is just the $(\ell+1)$-ary representation of the number $a' - a$. Similarly, if $a' \leq a$, then $(x_0, \dots, x_{n-1}) = (y_0 - e_0, \dots, y_{n-1} - e_{n-1})$, where $(e_0, e_1, \dots, e_{n-1})$ is the $(\ell+1)$-ary representation of $a - a'$.

For given ℓ, q and n, we define $LA_u(n,\ell)_q =$ the maximum size of an ℓ-UEC code, over the alphabet $[0, q-1]$, defined by a linear equation (4.5). Correspondingly we use $LA_a(n,\ell)_q$ for ℓ-AEC codes.

Theorem 13. *(i) For all n, q and ℓ, $LA_a(n,\ell)_q = LA_u(n,\ell)_q$.*
(ii) For all integers q, n and ℓ satisfying $q > \ell + 1$ we have

$$\frac{\ell}{q-1} \left(\frac{q}{\ell+1} \right)^n \leq LA_u(n,\ell)_q \leq \lceil \frac{q}{\ell+1} \rceil^{n-1}.$$

4.2 Construction of Optimal Codes

We call a VT-type ℓ-UEC code VT-type optimal (or shortly optimal) if it attains the upper bound in Theorem 13. Given integers $\ell \in [1, q-1]$, n, r we define

$$\mathcal{C}_n(r) = \left\{ (x_0, \dots, x_{n-1}) \in Q^n : \sum_{i=0}^{n-1} (\ell+1)^i x_i = \alpha S_n + r \right\}, \tag{4.6}$$

$$\text{where } S_n := \sum_{i=0}^{n-1} (\ell+1)^i = \frac{(\ell+1)^n - 1}{\ell}, \quad \text{and } \alpha := \lfloor \frac{q-1}{2} \rfloor. \tag{4.7}$$

It can be seen that $\mathcal{C}_n(r)$ is an ℓ-UEC code for all n and r.

We use the notation $\langle x \rangle_y$ to denote the integer in $[0, y-1]$ that is equivalent to x modulo y.

Theorem 14. *Let $u_1, u_2, \dots$ and $v_1, v_2, \dots$ be sequences of integers such that:*
(i) $0 \le u_1 + \alpha \le v_1 + \alpha \le q-1$,
and for each $n \ge 2$
(ii) $\lceil \frac{1}{\ell+1}(u_n + \alpha - (q-1)) \rceil \ge u_{n-1}$,
(iii) $\lfloor \frac{1}{\ell+1}(v_n + \alpha) \rfloor \le v_{n-1}$, *and*
(iv) $\ell+1$ divides q, or for each $r \in [u_n, v_n]$, $\langle \alpha + r \rangle_{\ell+1} < \langle q \rangle_{\ell+1}$.

Then for each $n \ge 1$ and $r \in [u_n, v_n]$ we have $|\mathcal{C}_n(r)| = \lceil \frac{q}{\ell+1} \rceil^{n-1}$.

Theorem 15. *Let ℓ and q be such that $\ell+1$ divides q. Let $u_1 = -\alpha$, $v_1 = \alpha$, and for $n \ge 2$, $u_n = (\ell+1)u_{n-1} + \alpha$ and $v_n = (\ell+1)v_{n-1} - \alpha$. In other words, for $n \ge 1, v_n = -u_n = \frac{\alpha}{\ell}((\ell-1)(\ell+1)^{n-1} + 1)$. Then for each $n \ge 1$ and $r \in [u_n, v_n]$, we have*

$$|\mathcal{C}_n(r)| = LA_u(n, \ell)_q = \left(\frac{q}{\ell+1} \right)^{n-1}.$$

Theorem 16. *Let $q = (b-1)(\ell+1) + d$, where the integers b, d and ℓ are such that $1 \le b-1 < d \le \ell$. Then for each n we can construct an ℓ-UEC code $\mathcal{C}_n(r)$ with*

$$|\mathcal{C}_n(r)| = LA_u(n, \ell)_q = b^{n-1} = \left\lceil \frac{q}{\ell+1} \right\rceil^{n-1}.$$

4.3 Open Problems

Give constructions for asymmetric/unidirectional codes, capable of correcting /detecting t errors of a given magnitude, with efficient coding and decoding schemes.

For practical application of those codes [22] (e.g. in multi-level flash memories), it is important to have efficient constructions of systematic codes (that is codes having systematic encoders), that are advantageous in high-speed memory architecture.

Give constructions of AEC/UEC-codes of a limited magnitude, correcting bursts of errors.

5 Parallel Error–Control Codes

In [11] Ahlswede, Balkenhol, and Cai introduced a new code concept for multiple-access channels (MAC) with a special error control mechanism. A communication channel consists of several sub–channels transmitting simultaneously and synchronously. The senders encode their messages into codewords of the same length over the same alphabet and transmit them in parallel. When an error occurs in a line at time T with a relatively high probability, an error also occurs in its neighbor lines. A parallel t-error correcting code is a code capable of correcting all t or less errors of this type. A parallel code is called independent, if the encoders proceed independently, that is, the code in this case is the Cartesian product of the codes used by the senders. As an example consider a parallel port of a computer device, where the message from the computer to the device is transmitted in parallel over a set of lines. A magnetic influence from outside produces errors during the transmission. However, the time instances when errors occur in the different lines are related. Thus we have a model for a coding problem for a MAC.

The model of parallel error correcting codes described above can be useful for the design of network error correcting codes in real networks. For instance, if we model a large link as several parallel links, an error of a link may cause the error for all associated links.

For blocklength n, messages are encoded by q-ary $r \times n$ matrices. In the channel considered in [11] the errors are of the additive type. To each row-vector in a code matrix M the same error vector e is added, that is, the $r \times n$ matrix E, called error matrix, with identical row vectors e is added.

In [9] we introduce a new model of a one-way channel, which is again based on parallel subchannels and again has the same error vectors, however, the errors are produced by the binary Z-channels (Boolean sums) now. We therefore call it *Parallel Error Z-channel* (PEZ-channel).

Recall that the binary Z-channel, has the property that only $0 \to 1$ (or $1 \to 0$) type of errors can occur during the transmission. This type of errors are called asymmetric. Here we consider errors of type $0 \to 1$.

In case errors are not correlated, but are produced letterwise again by Z-channels, we speak about the *Parallel Z-channel* (PZ-channel). We study it under the constraint: all letterwise errors occur in at most t columns.

A code $\mathcal{C}$, called $(r \times n)$-code, is a set of $r \times n$ $(0,1)$–matrices. We say that t parallel asymmetric errors have occurred in a sent matrix M, called *code matrix*,if in some t columns of M all zero entries turn into ones. The received word M' can be written as $M' = M \oplus E$, where the *error matrix* E is an $r \times n$ matrix with each column consisting of all ones or all zeros and $\oplus$ means the Boolean sum of $(0,1)$-matrices. The weight $w(E)$ is the number of nonzero columns in E.

We say that an $(r \times n)$–code $\mathcal{C}$ is capable of correcting t (parallel asymmetric) errors if any transmitted code matrix can be uniquely reconstructed at the receiving end in the presence of t or less errors. In other words, for every two codematrices M_1, M_2 and error matrices E_1, E_2 of weight not greater than t we have

$$M_1 \oplus E_1 \neq M_2 \oplus E_2. \tag{5.1}$$

We also say that $\mathcal{C}$ is capable of detecting t errors if

$$M_1 \oplus E \neq M_2 \tag{5.2}$$

holds for all E with $w(E) \leq t$. Such a code is called *t–parallel asymmetric error correcting/detecting code* (shortly *$(r \times n, t)$ PEZ-code*).

Similarly we define error correcting/detecting codes for the PZ-channel. The $0 \to 1$ errors can occur now in at most t columns. That is, an error E now is an $r \times n$ matrix of weight $w(E) \leq t$ (the weight of E is defined as above). Codes capable of correcting/detecting such type of errors are called here *$(r \times n, t)$ PZ-codes*. More precisely, a t error correcting (resp. detecting) $(r \times n)$ PZ-code is a code that satisfies the condition (5.1) (resp. condition (5.2)).

5.1 Error Correcting/Detecting Codes for *PEZ*-Channel

For an $r \times n$ $(0,1)$-matrix M the columns of M can be viewed as elements of the alphabet $Q = \{0, 1, \ldots, q-1\}$ $(q = 2^r)$ using an arbitrary one-to-one mapping $\varphi : \{0,1\}^r \to Q$. Thus any matrix M can be represented as an n-tuple $(a_1, \ldots, a_n) \in Q^n$. A natural way is to consider each column as the binary expansion of the corresponding number from Q. Our PEZ-channel can be illustrated now as a q-ary channel (with $q = 2^r$) called here q-ary Z-channel (shortly Z_q-channel) shown in Figure 2. In case $q = 2$ this is simply the Z-channel.

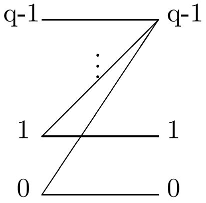


Fig. 2. q-ary Z-channel

Thus, the PEZ-channel is a special case of the Z_q-channel when $q = 2^r$. Therefore, in general it makes sense to study this channel for arbitrary q. The notion of t-error correcting/detecting codes is extended to any Z_q-channel in a natural way. Such codes are called here Z_q-code capable of correcting/detecting t errors.

Optimal error-detecting codes. Recall the notion of $S(n, q-1)$ introduced in Section 2 and let $W_i := |S_i(n, q-1)| = \binom{n}{i}(q-1)^{n-i}$, $i = 0, 1, \ldots, n$.

Theorem 17. *Given integers $n, a \geq 1$, and $1 \leq t < n$ we have*

(i) For arbitrary $a \in [0, t]$ the code $\mathcal{C}_a$ defined by

$$\mathcal{C}_a = \{x \in S_i(n, q-1) : i \equiv a \mod (t+1)\} \tag{5.3}$$

is a Z_q-code capable of detecting t errors.

(ii) The code C_{a^*} with $|\mathcal{C}_{a^*}| = \max\{|\mathcal{C}_a| : a \in [0,t]\}$ is an optimal t error detecting code.

Note that

$$|\mathcal{C}_{a^*}| = \max_{a\in[0,t]} \sum_{i\geq 0} W_{a+i(t+1)} > \frac{q^n}{t+1}. \tag{5.4}$$

In particular, for $W_k := \max\{W_i : 0 \leq i \leq n\}$ the theorem says that $S_k(n, q-1)$ is an optimal Z_q-code capable of detecting all errors.

Next we consider Error–correcting Z_q-codes.

Clearly any code capable of correcting t symmetric errors is a t error correcting Z_q-code. It is also not hard to see that a t error correcting Z_q-code is capable of detecting t or less symmetric errors. Therefore, an upper bound for a code with the minimum distance $d_H = t+1$ is a trivial upper bound for a t error correcting Z_q-code.

Let us, in particular, consider the case when $n \leq q+1$ and let $\mathcal{C}$ be a Z_q-code correcting t errors. The minimum Hamming distance of this code $d_H(\mathcal{C}) \geq t+1$ and the Singleton bound $|\mathcal{C}| \leq q^{n-t}$ (see [39]) is a trivial upper bound for $\mathcal{C}$. Note also that in case of prime power q we can use MDS codes (codes attaining the Singleton bound, see [39], Ch.11), with the minimum distance $d_H = 2t+1$ and size q^{n-2t}, as t error correcting Z_q-codes. However one can do better.

Consider in particular single-error correcting Z_q-codes. Then an MDS code with the minimum distance $d_H = 3$ has cardinality q^{n-2}. On the other hand the following parity check code has a greater size. Let us denote $Q^* = [0, q-2]$.

Proposition 7. *Given q the code $\mathcal{C} \subset Q^n$ defined by*

$$\mathcal{C} = \{(x_1, \ldots, x_n) \in Q^{*n} : \sum_{i=1}^{n} x_i \equiv a \mod q-1\} \tag{5.5}$$

is a single error correcting Z_q-code of cardinality $|\mathcal{C}| = (q-1)^{n-1}$.

One can extend the construction to t-error correcting Z_q-codes. It is sufficient to construct codes of length n and minimum distance $d_H(\mathcal{C}) = t+1$ over alphabet $Q^* = [0, q-2]$.

Remark. Such codes can be viewed as codes correcting *erasures* with the erasure symbol $q-1$. The *erasure channel*, in which each alphabet symbol is lost with a fixed probability (that is, turned into an erasure symbol "$*$ "), was introduced by P. Elias [24]. Erasure correcting codes (Fountain codes, LT codes etc.) are widely used for reliable networks (see e.g. [53]), and recently in Network Coding problems.

Formally, a t error correcting Z_q- code can be viewed as a code, capable of correcting t erasures, in which the erasure symbol can also be used for the transmission.

Thus, erasure correcting codes can be used as Z_q-codes. Note however that the size of a t–error correcting Z_q-code can be much larger than the size of a corresponding optimal erasure code over an alphabet of size $q-1$. To show that, we describe a more general construction of t–error correcting Z_q-codes.

Construction: Let C be a binary code of length capable of correcting t asymmetric errors. Let also $\mathcal{D} = \{D_m\};\ m = 1, \dots, n$ be a set of codes of length m capable of detecting t symmetric errors (i.e. a code with minimum Hamming distance $t+1$) over the alphabet Q^*. Note that some of D_m could be trivial codes containing only one codeword (by convention the minimum distance in a trivial code is ∞). Given a codeword $v^n \in C$ of Hamming weight $wt_H(v^n) = r$ let $\{i_1, \dots, i_r\}$ be nonzero coordinates of v^n and let $\{j_1, \dots, j_{n-r}\} = [1, n] \setminus \{i_1, \dots, i_r\}$ where $j_1 < \dots < j_{n-r}$. Define then $D(v^n) = \{(x_1, \dots, x_n) \in Q^n : x_{i_1} = \dots = x_{i_r} = q-1$ and $(x_{j_1}, \dots, x_{j_{n-r}}) \in D_{n-r}\}$.

We define now the code $\mathcal{C} = C \circ \mathcal{D}$ where

$$C \circ \mathcal{D} := \bigcup_{v^n \in \mathcal{C}} D(v^n). \tag{5.6}$$

Proposition 8. *Given integers $1 \leq t \leq n$ and $q > 2$ the code $\mathcal{C} = C \circ \mathcal{D}$ is a t error correcting Z_q- code of length n over alphabet $Q = [0, q-1]$.*

Notice that given n and q, the size of an optimal Z_q-code $C \circ \mathcal{D}$ capable of correcting t errors is greater than the size of an optimal code $\mathcal{C}'$ of length n, over an alphabet (say Q^*) of size $q-1$, capable of correcting t erasures. Indeed, let C (the code in our construction) contain the all zero vector 0^n. Then clearly $\mathcal{C}' \subset C \circ \mathcal{D}$.

Next we apply the described approach for construction of single error correcting Z_q-codes for arbitrary n and q. We use Varshamov-Tennengolts codes (VT codes) for construction of q–ary single–error correcting Z_q-codes.

Given integers $n \geq 1$ and $q > 2$ $(Q = [0, q-1],\ Q^* = [0, q-2])$ let $C(n, a)$ be a VT code.

For $m = 1, \dots, n$ and $\alpha \in Q^*$ we define now $\mathcal{D} = \{D_1(\alpha), \dots, D_n(\alpha)\}$ with

$$D_m(\alpha) := \{x^m \in Q^{*m} : \sum_{i=1}^{m} x_i \equiv \alpha \mod q-1\}. \tag{5.7}$$

Each code $D_m(\alpha)$ has size $|D_m(\alpha)| = (q-1)^{m-1}$ and minimum Hamming distance 2 $(m \in [1, n]\ ,\ \alpha \in [0, q-2])$. In view of Proposition 7 the code

$$\mathcal{C}(n, a, \alpha) := C(n, a) \circ \mathcal{D}$$

is a single-error correcting Z_q-code.

Let $A_0(n, a), A_1(n, a), \dots, A_n(n, a)$ be the weight distribution of $C(n, a)$, that is $A_i(n, a) := \#$ {codewords of Hamming weight i}.

Then it can be easily seen that $|\mathcal{C}(n, a, \alpha)| = \sum_{i=0}^{n-1} A_i(n, a) \cdot (q-1)^{n-i-1} + A_n(n, a)$. Since $|\mathcal{C}(n, a, \alpha)| = |\mathcal{C}(n, a, 0)|$ we denote $\mathcal{C}(n, a) = \mathcal{C}(n, a, 0)$. Thus we have the following

Theorem 18. *For integers* $0 \leq a \leq n$ *and* $q \geq 3$ *the code* $\mathcal{C}(n,a)$ *is a* q*–ary single–error correcting* Z_q*-code with*

$$|\mathcal{C}(n,a)| = \sum_{i=0}^{n-1} A_i(n,a) \cdot (q-1)^{n-i-1} + A_n(n,a). \tag{5.8}$$

Example. $n = 8,\ q = 4\ (r = 2), a = 0.$
Let A_i denote the number of codewords of weight i in the VT code $C(8,0)$. We have $A_0 = A_8 = 1,\ A_1 = A_7 = 0,\ A_2 = A_6 = 4,\ A_3 = A_5 = 6,\ A_4 = 8.$

Our construction gives us a single–error correcting $(2 \times 8, 1)$ Z_q-code $\mathcal{C}(8,0)$ with $|\mathcal{C}(8,0)| = A_0 \cdot 3^7 + A_2 \cdot 3^5 + A_3 \cdot 3^4 + A_4 \cdot 3^3 + A_5 \cdot 3^2 + A_6 \cdot 3 + A_8 = 3^7 + 4 \cdot 3^5 + 6 \cdot 3^4 + 8 \cdot 3^3 + 6 \cdot 3^2 + 4 \cdot 3 + 1 = 3928.$

Note that the size of a single symmetric error correcting code of length 8 (over an alphabet of size 4) is upper bounded (Hamming bound) by $\lfloor 2^{16}/(3 \cdot 8 + 1) \rfloor = 2621$.

Next we give an upper bound for a single–error correcting Z_q–code.

Theorem 19. *Let* $\mathcal{C}(n)_q$ *be a single–error correcting* Z_q*-code of length* n*. Then*

$$|\mathcal{C}(n)_q| < \sum_{k=0}^{n-1} \frac{\binom{n}{k}(q-1)^{n-k-1}}{k+1}. \tag{5.9}$$

5.2 Codes for PZ-Channels

Let $\mathcal{M}(r \times n)$ be the set of all $r \times n$ $(0,1)$–matrices. A subset $\mathcal{C} \subset \mathcal{M}(r \times n)$ is a t-error correcting/detecting PZ-code if $\mathcal{C}$ is capable of correcting/detecting all asymmetric errors in t or less columns. We call such codes for short $(r \times n, t)$–codes.

Note that any t-error correcting/detecting PZ-code $\mathcal{C} \subset \mathcal{M}(r \times n)$ is also a t–error correcting/detecting PEZ code. We discuss first the error detection problem.

For $A \in \mathcal{M}(r \times n)$ the Hamming weight $w_H(A)$ is the number of nonzero entries in A.

Theorem 20. *Given integers* $1 \leq t \leq n$*,* $1 \leq r$

$$\mathcal{A} := \left\{ A \in \mathcal{M}(r \times n) : wt_H(A) \equiv \lfloor \frac{rn}{2} \rfloor \bmod (tr+1) \right\} \tag{5.10}$$

is a t*-error detecting* PZ*-code.*

Note that the code $\mathcal{A}$ defined by (5.10) is optimal for $r = 1$, however, this is not the case in general.

Theorem 21. *Given integers* $1 < r$ *and* $1 \leq t < n$*, let* $\mathcal{A}(r \times n, t)$ *be an optimal* t*-error detecting* PZ *code. Then*

$$\frac{2^{rn}}{tr+1} \leq |\mathcal{A}(r \times n, t)| \leq \frac{2^{rn}}{\sqrt{tr}}. \tag{5.11}$$

The lower bound in (5.11) follows from the code construction in Theorem 20.

We consider now the error correction problem for the simplest case $t = 1$.

Every matrix $M \subset \mathcal{M}(r \times n)$, with columns $\bar{b}_1, \ldots, \bar{b}_n$, is associated with the sequence $(b_1, \ldots, b_n)$ where $\bar{b}_i$ $(i = 1, \ldots, n)$ is the binary representation of b_i. For a subset $\mathcal{S} \subset Q^n$, $Q := [0, 2^r - 1]$ we denote by $\mathcal{S}(r \times n) \subset \mathcal{M}$ the set of matrices corresponding to the elements of $\mathcal{S}$.

We say that there exists a k–factorization of $\mathbb{Z}_m^*$ ($\mathbb{Z}_m^* := \mathbb{Z}_m \setminus \{0\}$) if there exists a subset $A \subset \mathbb{Z}_m^*$ such that each element of $\mathbb{Z}_m^*$ can be uniquely represented as a product $i \cdot a$ where $i \in \{1, \ldots, k\}$ and $a \in A$.

Theorem 22. *Given integers $n, r \geq 2$ let $m := n(2^r - 1) + 1$ and let there exist a $(2^r - 1)$–factorization of $\mathbb{Z}_m^*$ by a subset $A = \{a_1, \ldots, a_n\}$. For $a \in \mathbb{Z}_m$ let $\mathcal{B}(r \times n) \subset Q^n$ be defined by*

$$\mathcal{B}(r \times n) = \{(x_1, \ldots, x_n) \in Q^n : \sum_{i=1}^{n} a_i x_i \equiv a \mod m\}. \tag{5.12}$$

Then $\mathcal{B}(r \times n)$ is a single-error correcting PZ-code with

$$|\mathcal{B}(r \times n)| \geq \frac{2^{rn}}{n(2^r - 1) + 1}. \tag{5.13}$$

Example. Let $n = 12$, $r = 2$, and hence $n(2^r - 1) + 1 = 37$. One can check that there exists a 3–factorization of $\mathbb{Z}_{37}^*$ by the set $A = \{2, 9, 12, 15, 16, 17, 20, 21, 22, 25, 28, 35\}$. That is $\mathbb{Z}_{37}^* = A \cup 2A \cup 3A$ where $iA := \{ia \mod 37 : a \in A\}$, $i = 2, 3$. Therefore, the code $\mathcal{B}(2 \times 12)$ defined by (5.12) is a single-error correcting PZ-code with cardinality $|\mathcal{B}(2 \times 12)| \geq 4^{12}/37$ exceeding the Hamming bound for a quaternary single symmetric error correcting code of length 12.

We describe now a construction of single-error correcting PZ-codes with a very simple decoding algorithm.

Code construction. For integers $1 < r \leq n$, let $\mathcal{E}(r \times n)$ denote the set of all $r \times n$ $(0, 1)$-matrices with even row weights. Thus $|\mathcal{E}(r \times n)| = 2^{(n-1)r}$. For an $r \times n$ $(0, 1)$-matrix M let $h_i(M)$ denote the Hamming weight of its i-th column. Let also p be the smallest prime such that $p \geq n + 1$. We define the code $\mathcal{C}(r \times n)$ as follows.

$$\mathcal{C}(r \times n) = \{M \subset \mathcal{E}(r \times n) : \sum_{i=1}^{n} i \cdot h_i(M) \equiv a \mod p\}. \tag{5.14}$$

Theorem 23. *(i) $\mathcal{C}(r \times n)$ is capable of correcting all asymmetric errors in a single column.*

(ii) There exists $0 \leq a \leq p - 1$ such that $|\mathcal{C}(r \times n)| \geq 2^{(n-1)r}/p$.

Decoding algorithm

For a received word $M' \in \mathcal{M}(r \times n)$

Step 1. Determine the column vector
$(\varepsilon_1, \dots, \varepsilon_r)^T := M' \cdot (1, \dots, 1)^T \mod 2$.
Step 2. Compute $t := w_H(\varepsilon_1, \dots, \varepsilon_r)$.
If $t = 0$ then M' is a code matrix, otherwise
Step 3. Compute $b := \sum_{i=1}^{n} i \cdot h_i(M') \mod p$.
Step 4. Compute $k := \frac{b-a}{t} \mod p$.
Step 5. Evaluate the error matrix $E \in \mathcal{M}(r \times n)$,
with the k-th column $(\varepsilon_1, \dots, \varepsilon_r)^T$ and with zero entries elsewhere.
Setep 6. Determine the transmitted code matrix $M = M' - E$.

5.3 Open Problems

A challenging combinatorial optimization problem is construction of optimal or near optimal t–error detecting codes for PZ–channels (even for $t = 1$).

Constructions of "good" t–error correcting codes (for both channels) with efficient decoding algorithms is another problem for further research.

We considered only errors occurring in a restricted set of columns. Consider codes for correction/detection clusters of errors.

6 Further Work and Reports

Our work not reported here concerns the subjects: Network coding, Identification entropy, Weighted constrained error-correction and will be briefly discussed below to motivate the reader to read our contributions, which can be found in the books "General Theory of Information Transfer and Combinatorics" (Eds. R. Ahlswede et al.), Lecture Notes in Computer Science, Vol. 4123, Springer Verlag, 2006 and "Lectures on Advances in Combinatorics" (R. Ahlswede and V. Blinovsky), Universitext, Springer Verlag, 2008, the Special Issue "General Theory of Information Transfer and Combinatorics" (Eds. R. Ahlswede et al.) of Discrete Applied Mathematics, Vol. 156, No. 9, 2008, and in the 2006 Shannon Lecture " Towards a General Theory of Information Transfer" (R. Ahlswede), Shannon Lecture at ISIT in Seattle 13th July 2006, IEEE Inform. Theory Society Newsletter, Vol. 57, No. 3, 6-28, 2007.

6.1 Network Coding

Network coding was introduced by Ahlswede, Cai, Li, and Yeung [13] to improve network throuput and performance. It has emerged as a new paradigm that has influenced Information and Coding Theory, Networking, Wireless Communications, Computer Science, Graph Theory etc. (see Network Coding Homepage http://www.ifp.uiuc.edu/ koetter/NWC/) The basic idea of Network Coding is to allow the intermediate nodes to process the received information before forwarding them.

Ahlswede *et al.*[13] showed that by Network Coding one can achieve the multicast capacity in information networks with a single source. Li *et al.* [38] showed that Linear Coding suffices to achieve Koetter and Medard [34] gave an algebraic characterization of the linear coding schemes that achieve the capacity. Jaggi, Sanders *et al.* [28] gave a polynomial time algorithm to construct linear codes for single source multicast. The existence of such algorithms is remarkable since the maximum rate without coding can be much smaller and finding the maximum rate routing solution is NP-hard.

Network coding is believed to be highly applicable to communication through real networks, the primary example being the Internet (the most widely known application is the Avalanche program by Microsoft for file distribution protocols). In addition to *throughput gain*, many other benefits such as minimization of *delay*, minimization of *energy per bit*, *robustness, adaptability etc.* of Network Coding have been discovered during the last years.

Research in Network Coding is growing fast (more than 250 papers appeared since 2002). Microsoft, IBM and other companies have research teams who are investigating this new field. A few American universities (Princeton, MIT, Caltech and Berkeley) have also established research groups in Network Coding.
The holy grail in Network Coding is to plan and organize (in an automated fashion) network flow (that is to allow to utilize network coding) in a feasible manner.

In Riis and Ahlswede [50] new links between Network Coding and Combinatorics are established. It is shown that the task of designing efficient strategies for information network flow (Network Coding) is closely linked to designing error correcting codes. This link is surprising since it appears even in networks where transmission mistakes never happen! Recall that traditionally error correction is mainly used to reconstruct messages that have been scrambled due to unknown (random) errors. We use error correcting codes when channels are assumed to be error-free. Thus error correcting codes can be used to solve network flow problems even in a setting where errors are assumed to be insignificant or irrelevant.

6.2 Identification Entropy

Classical transmission concerns the question "How many messages can we transmit over a noisy channel?" One tries to give an answer to the question "What is the actual message from $\mathcal{M} = \{1, \dots, M\}$?" On the other hand in *Identification* it is asked "How many possible messages can the receiver of a noisy channel identify?" One tries to give an answer to the question "Is the actual message i?". Here i can be any member of the set of possible messages $\mathcal{N} = \{1, 2, \dots, N\}$.

On the Source Coding side Ahlswede and Cai [12] introduced the concept of identification entropy, namely the function

$$H_{I,q}(P) = \frac{q}{q-1}\left(1 - \sum_{u\in\mathcal{U}} P_u^2\right).$$

We proved that $L_{\mathcal{C}}(P,P) = \sum\limits_{u\in\mathcal{U}} P_u \; L_{\mathcal{C}}(P,u) \geq H_{I,q}(P)$ and thus also that

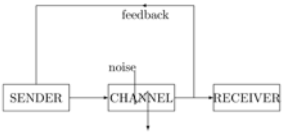
$$L(P) = \min_{\mathcal{C}} \max_{u\in\mathcal{U}} L_{\mathcal{C}}(P,u) \geq H_{I,q}(P)$$

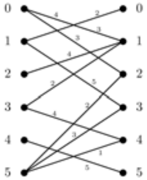
and related upper bounds, which demonstrate the operational significance of identification entropy in *Noiseless Source Coding* similar as *Boltzmann/Shannon entropy* does in *Noiseless Data Compression.*

This theory initiated other research areas like Common Randomness, Authentication in Cryptology, and Alarm Systems. It also led to the discovery of new methods which became fruitful also for the classical theory of transmission, for instance in studies of robustness like arbitrarily varying channels, optimal coding procedures in case o f complete feedback, novel approximation problems for output statistics and generation of common randomness, the key issue in Cryptology.

6.3 Weighted Constrained Error-Correction

The Rényi-Berlekamp-Ulam game is a model for determining the minimum number of queries to find an unknown member in a finite set when up to a finite number of the answers may be erroneous. Questions with q many possible answers are allowed. Errors in the answer are constrained by a bipartite graph with edges weighted by $0, 1, 2, \ldots$ (the "channel").





The channel Γ is an arbitrary, though fixed, assignment stipulating the cost of the different possible errors, i.e., of each answer $j \neq i$ when the correct answer is i by $\Gamma(i, j)$. It is also assumed that a maximum cost e (sum of the cost of all wrong answers) can be afforded by the responder. We provided a tight asymptotic estimate [14] for the number of questions needed to solve this problem.

References

1. Ahlswede, R., Aydinian, H.: Sparse asymmetric connectors in communication networks. In: Ahlswede, R., Bäumer, L., Cai, N., Aydinian, H., Blinovsky, V., Deppe, C., Mashurian, H. (eds.) General Theory of Information Transfer and Combinatorics. LNCS, vol. 4123, pp. 1056–1062. Springer, Heidelberg (2006)
2. Ahlswede, R., Aydinian, H.: Construction of asymmetric connectors of depth two. Special Issue in Honor of van Lint, J.H. of J. Combinatorial Theory, Series A 113(8), 1614–1620 (2006)
3. Ahlswede, R., Aydinian, H.: Diagnosability of large multiprocessor systems. In: 2nd COMBSTRU workshop 2004, Venice, September 20-22 (2004)
4. Ahlswede, R., Aydinian, H.: On diagnosability of large multiprocessor networks. Electronic Notes in Discrete Mathematics 21, 101–104 (2005); Discrete Applied Math. 156(18), 3430–3442 (2008)

5. Ahlswede, R., Aydinian, H., Khachatrian, L.H.: Unidirectional error control codes and related combinatorial problems. In: Proceedings of Eight Intern. Workshop on Algebraic and Combinatorial Coding Theory, Tsarskoe Selo, Russia, September 6-9, pp. 8–14 (2002)
6. Ahlswede, R., Aydinian, H.: On t/s- diagnosability of multiprocessor systems. In: Final conference: General Theory of Information Transfer and Combinatorics, Center of Interdisciplinary Research (ZIF), Bielefeld, April 26-30 (2004)
7. Ahlswede, R., Aydinian, H.: An extremal problem on graphs. In: Proceedings of Workshop on graphs and combinatorial optimization, Lambrecht, Germany, June 6-9 (2006)
8. Ahlswede, R., Aydinian, H., Khachatrian, L.H., Tolhuizen, L.M.G.M.: On q–ary codes correcting all unidirectional errors of a limited magnitude. In: Proc. Ninth Intern. workshop on Algebraic and Combin. Coding Theory, Kranevo, Bulgaria, June 19–25, pp. 20–26 (2004) (to appear in a special issue dedicated to Varshamov), `http://arxiv.org/pdf/cs.IT/0607132`
9. Ahlswede, R., Aydinian, H.: Error correcting codes for parallel asymmetric channels. In: Proceedings of International Symposium on Information Theory ISIT 2006, Seattle, July 6-12 (2006)
10. Ahlswede, R., Aydinian, H.: Error control codes for parallel asymmetric channels. IEEE Trans. Inform. Theory 54(2), 831–836 (2008)
11. Ahlswede, R., Balkenhol, B., Cai, N.: Parallel error correcting codes. IEEE Trans. Inform. Theory 48(4), 959–962 (2002)
12. Ahlswede, R., Cai, N.: An interpretation of identification entropy. IEEE Trans. Inform. Theory 52(9), 4198–4207 (2006)
13. Ahlswede, R., Cai, N., Li, S.Y.R., Yeung, R.W.: Network information flow. IEEE Trans. Inform. Theory 46(4), 1204–1216 (2000)
14. Ahlswede, R., Cicalese, F., Deppe, C.: Searching with lies under error transition cost constraints. General Theory of Information Transfer and Combinatorics, Special Issue of Discrete Applied Mathematics 156(9), 1444–1460 (2008)
15. Ahlswede, R., Koschnick, K.U.: Note on an extremal problem arising for unreliable networks in parallel computing. Discrete Math. 47, 137–152 (1983)
16. Baltz, A., Jäger, G., Srivastav, A.: Constructions of sparse asymmetric connectors with number theoretic methods. Networks 45(3), 1–6 (2005)
17. Barborak, M., Malek, M., Dahbura, A.: The consensus problem in fault-tolerant computing. ACM Computing Surveys 25(2), 171–220 (1993)
18. Beneš, V.E.: Optimal rearrangeable multistage connecting networks. Bell System Tech. J. 43, 1641–1656 (1964)
19. Blaum, M. (ed.): Codes for Detecting and Correcting Unidirectional Errors, IEEE Computer Society Press Reprint Collections. IEEE Computer Society Press, Los Alamitos (1993)
20. Bollobas, B.: Extremal Graph Theory. Academic Press, London (1978)
21. Caruso, A., Chessa, S., Maestrini, P., Santi, P.: Diagnosability of regular systems. J. Algorithms 45, 126–143 (2002)
22. Cassuto, Y., Schwartz, M., Bohossian, V., Bruck, J.: Codes for multilevel flash memories: correcting asymmetric limited magnitude errors. In: International Symposium on Information theory ISIT 2007, Nice, France, June 24- 29, pp. 1176–1180 (2007)
23. Clos, C.: A study of non–blocking switching networks. Bell System Tech. J. 32, 406–424 (1953)
24. Elias, P.: The noisy channel coding theorem for erasure channels. Amer. Math. Monthly 81, 853–862 (1974)

25. Engel, K.: Sperner Theory. Cambridge University Press, Cambridge (1997)
26. Feldman, P., Friedman, J., Pippenger, N.: Wide-sense nonblocking networks. SIAM J. Discr. Math. 1, 158–173 (1988)
27. Friedman, A.D.: A new measure of digital system diagnosis. In: Proc. Fifth Intern. Symp. Fault Tolerant Computing, pp. 167–170 (1975)
28. Jaggi, S., Sanders, P., Chou, P.A., Effros, M., Egner, S., Jain, K., Tolhuizen, L.M.G.M.: Polynomial time algorithms for multicast network code construction. IEEE Trans. Inform. Theory 51(6), 1973–1982 (2005)
29. Harper, L.H.: Optimal numberings and isoperimetric problems on graphs. J. Combin. Theory 1, 385–395 (1966)
30. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bulletin AMS 43(4), 435–561 (2006)
31. Hwang, F.K., Richards, G.W.: A two–stage rearrangeable broadcast switching network. IEEE Trans. on Communications 33, 1025–1035 (1985)
32. Kavianpour, A., Kim, K.H.: Diagnosabilities of hypercubes under the pessimistic one-step diagnosis strategy. IEEE Trans. Comput. 40(2), 233–237 (1991)
33. Khanna, S., Fuchs, W.K.: A graph partitioning approach to sequantial diagnosis. IEEE Trans. Comput. 46(1), 39–47 (1996)
34. Koetter, R., Medard, M.: An algebraic approach to network coding. IEEE/ACM Trans. Netw. 11(5), 782–795 (2003)
35. Kløve, T.: Error correcting codes for the asymmetric channel, Report, Dept. of Math. Univ. of Bergen (1981) (with updated bibliography in 1995)
36. Kövari, T., Sós, V.T., Turán, P.: On a problem of K. Zarankiewicz. Colloquium Math. 3, 50–57 (1954)
37. Leeb, K.: Salami- Taktik beim Quader-Packen, Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung. Universität Erlangen 11(5), 1–15 (1978)
38. Li, S.-Y., Yeung, R.W., Cai, N.: Linear network coding. IEEE Trans. Inf. Theory 49(2), 371–381 (2003)
39. MacWilliams, F.J., Sloane, N.J.: The Theory of Error-Correcting Codes. North-Holland, New York (1988)
40. Oruc, A.Y.: A study of permutation networks: some generalizations and tradeoffs. J. of Parallel and Distributed Computing 22, 359–366 (1994)
41. Pippenger, N.: Communication Networks. In: Handbook of Theoretical Computer Science. Elsevier, Amsterdam (1990)
42. Pippenger, N., Yao, A.C.: On rearangeable networks with limited depth. SIAM J. Algebraic Discrete Methods 3, 411–417 (1982)
43. Pippenger, N.: On rearangeable and nonblocking switching networks. J. Comput. System Sci. 17, 145–162 (1987)
44. Preparata, F.P., Metze, G., Chien, R.T.: On the connection assignment problem of diagnosable systems. IEEE Trans. Comput. 16(12), 848–854 (1967)
45. Sanders, P., Egner, S., Tolhuizen, L.: Polynomial time algorithms for network information flow. In: 15th ACM Symposium on Parallel Algorithms and Architectures, pp. 286–294 (2003)
46. Santi, P., Chessa, S.: Reducing the number of sequential diagnosis iterations in hypercubes. IEEE Trans. Comput. 53, 89–92 (2004)
47. Shannon, C.E.: Memory requirements in a telephone exchange. Bell System Tech. J. 29, 343–349 (1950)
48. Slepian, D.: Two theorems on a particular crossbar switching network (1952) (unpublished manuscript)

49. Raghavan, V., Tripathi, A.: Sequential diagnosability is co-NP complete. IEEE Trans. Comput. 40(5), 584–595 (1991)
50. Riis, S., Ahlswede, R.: Problems in Network coding and error correcting codes, NETCOD 2005 (The First Workshop on Network Coding Theory and Applications). In: Ahlswede, R., Bäumer, L., Cai, N., Aydinian, H., Blinovsky, V., Deppe, C., Mashurian, H. (eds.) General Theory of Information Transfer and Combinatorics. LNCS, vol. 4123, pp. 861–897. Springer, Heidelberg (2006)
51. Varshamov, R.R., Tennengolts, G.M.: A code which corrects single asymmetric errors (in Russian). Avtomat. Telemeh. 26, 286–292 (1965) (transl: Automat. and Remote Contr. 26, 286–290, 1965)
52. Wicker, B., Bhargava, V.K. (eds.): Reed-Solomon Codes and their Applications. IEEE Press, New York (1994)
53. Wickler, S.B., Kim, S.: Fundamentals of Codes, Graphs, and Iterative Decoding. Kluwer Academic Publ., Norwell (2003)
54. Wigderson, A., Zuckerman, D.: Expanders that beat the eigenvalue bound: explicit construction and applications. Combinatorica 19(1), 125–138 (1999)
55. Yamada, T., Otsuka, T., Watanabe, A., Ueno, S.: On sequential diagnosis of multiprocessor systems. Discrete Appl. Math. 146(3), 311–342 (2005)

Resource Management in Large Networks

Susanne Albers

Department of Computer Science, University of Freiburg,
Georges Köhler Allee 79, 79110 Freiburg, Germany
salbers@informatik.uni-freiburg.de

Abstract. This survey article revisits resource management problems arising in large networks. More specifically, we focus on two fundamental memory management problems. In the first part of this paper we study buffer management in network routers and switches. We present various online strategies and report on their competitiveness and experimentally observed performance. The second part of the survey addresses web caching where a limited reordering of requests is allowed. We present both online and offline strategies.

1 Introduction

With the advent of the Internet algorithmic problems in large networks have received considerable research interest. The studies address optimization problems arising in hardware components of networks as well as computational issues related to network protocols. Within the research project "Resource management in large networks" we have investigated a variety of problems such as memory management [1,4,5,21], hotlink assignment [19] game theoretic issues of network design [2,3] and routing [16]. In this survey we focus on two fundamental memory management problems. These include buffer management problems in network switches and caching problems in web servers or browsers.

The general goal is to design and analyze algorithms having a provably good performance. Many optimization problems are NP-hard and hence the efficient computation of optimal solutions, most likely, is impossible. For this reason one resorts to approximations. Suppose that we consider a minimization problem. A polynomial time algorithm A achieves an *approximation ratio* of c if there exists a constant b such that $A(I) \leq c \cdot OPT(I) + b$ holds for all inputs I. Here $A(I)$ and $OPT(I)$ denote the objective function values of A and of an optimal solution OPT on I. The constant b must be independent of the input I. In the literature, our definition of an approximation guarantee is also referred to as *asymptotic approximation ratio*. A stricter definition requires $A(I) \leq c \cdot OPT(I)$, for all inputs I. In this paper we adopt the more relaxed variant, allowing an additional b in the cost analysis.

Furthermore, many computational problems are inherently *online*, i.e. the input arrives incrementally over time. Whenever a new input portion arrives, an algorithm must react and compute partial output not knowing future input.

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 227–246, 2009.

In this context, for a given minimization problem, an online algorithm A is *c-competitive* [22] if there exists a constant b such that, for all input sequences I, we have $A(I) \leq c \cdot A(I) + b$. Again, the constant b must be independent of the input.

In this survey article we first present the most important results we have achieved for buffer management problems in network switches. We will discuss theoretical as well as experimental results. Then we will review our contributions for web caching.

2 Packet Buffering

The performance of high-speed networks critically depends on switches that route data packets arriving at the input ports to the appropriate output ports so that the packets can reach their correct destination in the network. To reduce packet loss when the traffic is bursty, ports are equipped with buffers where packets can be stored temporarily. However these buffers are of limited capacity so that effective buffer management strategies are important to maximize the throughput of a switch. While packet buffering strategies have been investigated in the applied computer science and, in particular, networking communities for many years, only a seminal paper by Kesselman et al. [20] from 2001 has initiated profound algorithmic studies.

Consider the basic architecture of a Combined Input/Output Queued (CIOQ) switch with m input and m output ports. Figure 1 depicts a sample structure where $m = 3$. In such a structure, each input port i, $1 \leq i \leq m$, maintains for each output port j, $1 \leq j \leq m$, a separate queue Q_{ij} storing those packets that arrive at input i and are meant for output j. Additionally, each output port j, $1 \leq j \leq m$, maintains a queue Q'_j that stores packets that actually have to be transmitted via that port. In each time step new packets arrive at the input ports and are enqueued at the corresponding buffers Q_{ij}. Then in a series of, say, s rounds packets residing in the queues Q_{ij} are forwarded to the desired buffers Q'_j. This is done using a switch fabrique. The parameter s is called the switch speed. The forwarding step involves some constraints. In each round, for each input i, only one packet from among those located at the heads of Q_{ij}, $1 \leq j \leq m$, may be forwarded to the desired output queue. Furthermore, in each round, for each j, only one packet from the heads of Q_{ij}, $1 \leq i \leq m$, may be appended at Q'_j. Hence, intuitively, in each round each input port may forward only one packet from its m buffers. Moreover, each output port may receive only one packet from the corresponding m input queues. These restrictions motivate the following multi-buffer problem that we have studied extensively in our project.

Formally, we are given m buffers, each of which can simultaneously store up to B data packets. The buffers are organized as queues. In any time step, in a packet arrival phase, new packets may arrive at the buffers and can be appended to the queues if space permits. Suppose that buffer i currently stores b_i packets and that a_i new packets arrive there. If $b_i + a_i \leq B$, then all new packets can be

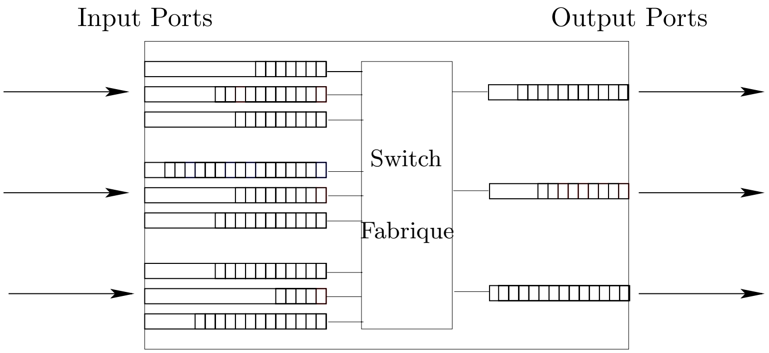


Fig. 1. A CIOQ switch with three input and three output ports

accepted; otherwise a_i+b_i-B packets must be dropped. Furthermore, in any time step, in a transmission phase, an algorithm can select one non-empty buffer and transmit the packet at the head of the queue to the output. We assume w.l.o.g. that the packet arrival phase precedes the transmission phase. The goal is to maximize the throughput, which is the total number of successfully transmitted packets. Obviously, the m buffers correspond to the queues Q_{ij}, for any fixed input $1 \leq i \leq m$ or for any fixed output $1 \leq j \leq m$. We remark that we assume that all packets have the same value, i.e. they are equally important. This is a reasonable assumption as most current networks, in particular IP networks, treat packets from different data streams equally at intermediate switches.

In the following we first present various algorithms for the above multi-buffer problem. We are mainly interested in online strategies and report on their competitive performance. Then we present an experimental study of the proposed algorithms.

2.1 Algorithms and Competitive Performance

A simple observation shows that any *work conserving* algorithm A, which just serves any non-empty queue, is 2-competitive: Partition σ into subsequences σ_ℓ such that A's buffers are empty at the end of each σ_ℓ. W.l.o.g. we postpone the beginning of $\sigma_{\ell+1}$ until OPT has emptied its buffers, too. Let T be the length of σ_ℓ, i.e. the number of time steps until A's buffers are empty. If OPT buffers b_i packets in queue i at the end of σ_ℓ, then at least b_i packets must have arrived there in σ_ℓ. Algorithm A has transmitted T packets, where $T \geq \sum_{i=1}^{m} b_i$, while OPT delivers at most $T + \sum_{i=1}^{m} b_i$ packets.

Prior to our work, Azar and Richter [7] presented a general technique that transforms a c-competitive algorithm for a single buffer problem into a $2c$-competitive algorithm for a multi-buffer problem. In the unit-value setting, this yields another 2-competitive strategy. However, the technique involves expensive simulations and does not seem to be applicable in practice.

In our contribution [5] we study *Greedy* algorithms, which represent the most simple and natural buffering strategies. In the following let the *load* of a queue be the number of packets currently stored in it.

> **Algorithm Greedy.** In each time step serve a queue currently having the maximum load.

From a practical point of view *Greedy* is very interesting because it is fast and uses little extra memory. More precisely, the algorithm just has to determine the most loaded queue and, regarding memory, only has to store its index. Serving the longest queue is a very reasonable strategy to avoid packet loss if future packet arrival patterns are unknown.

In [5] we first settle the exact performance of all *Greedy* strategies. Implementations of *Greedy* can differ in the way how ties are broken when several queues store a maximum number of packets. It turns out that this does not affect the competitive performance. As *Greedy*, obviously, is a work conserving algorithm, the following theorem is immediate.

Theorem 1. [5] *Greedy is* 2*-competitive, no matter how ties are broken.*

Unfortunately, *Greedy* is not better than 2-competitive as the lower bound of $2 - 1/B$ stated in the next theorem can be arbitrarily close to 2.

Theorem 2. [5] *For any B, the competitive ratio of Greedy is not smaller than $2 - 1/B$, no matter how ties are broken.*

Thus, *Greedy* is exactly 2-competitive, for arbitrary buffer sizes B. Based on the fact that *Greedy*'s competitive performance is not better than that of arbitrary work conserving strategies, an interesting problem is to design improved policies. The first deterministic online algorithm beating the bound of 2 was our *Semi Greedy* strategy [5]. The algorithm deviates from standard *Greedy* when all the queues are lightly populated, i.e. they store at most $B/2$ packets, and the current risk of packet loss is low. In this situation the algorithm preferably serves queues that have never experienced packet loss in the past.

> **Algorithm Semi Greedy.** In each time step execute the first of the following three rules that applies to the current buffer configuration. (1) If there is a queue buffering more than $\lfloor B/2 \rfloor$ packets, serve the queue currently having the maximum load. (2) If there is a queue the hitherto maximum load of which is less than B, then among these queues serve the one currently having the maximum load. (3) Serve the queue currently having the maximum load. In each of the three rules, ties are broken by choosing the queue with the smallest index. Furthermore, whenever all queues become empty, the hitherto maximum load is reset to 0 for all queues.

Theorem 3. [5] *Semi Greedy achieves a competitive ratio of* 17/9*.*

The ratio of 17/9 is approximately 1.89.

Subsequent to our work Azar and Litichevskey [6] devised a deterministic *Waterlevel* algorithm that achieves a competitive ratio of $\frac{e}{e-1}(1 + \frac{\lfloor H_m+1 \rfloor}{B})$. Here $H_m = \sum_{i=1}^{m} 1/i$ denotes the m-th Harmonic number. *Waterlevel* is quite involved and we only present a high-level description here. The main idea of *Waterlevel* is to compute an online matching, mapping time steps to data packets. If time step t is mapped to packet p, then p is served during that time. The initial matching is fractional in that a time step may be mapped to varying portions of several packets. Here the goal is to serve available packets as evenly as possible. In a cascade of algorithms, the fractional service schedule is then converted into a feasible integral schedule. Details can be found in [6].

The competitive ratio of *Waterlevel* is optimal as $B \to \infty$ because in [5] we have also established the following lower bound on the performance of deterministic strategies.

Theorem 4. [5] *The competitive ratio of any deterministic online algorithm is at least* $e/(e-1)$.

The ratio of $e/(e-1)$ is approximately 1.58.

We give a final deterministic online algorithm, called *HSFOD*, that is only 2-competitive but exhibits an excellent performance in practice. We developed this strategy during an experimental study [4], presented in the next Section 2.2, in which we have tested all of the above deterministic online algorithms as well as the randomized strategies described in the following paragraphs. *HSFOD* mimics an optimal offline algorithm, named *SFOD*, which we had developed in the initial paper [5]. More specifically, *SFOD* at any time transmits a packet from a non-empty queue that would overflow earliest in the future if no queues were served. This strategy, working offline as future packet arrivals must be known, achieves an optimal throughput on any input sequence. *HSFOD* (*Heuristic SFOD*) estimates future packet arrival rates by keeping track of past arrival patterns. Based on these arrival rates, the algorithm mimics *SFOD*.

> **Algorithm HSFOD.** The algorithm maintains packet arrival rates at the m input ports. These rates are updated after each packet arrival phase. For $1 \leq i \leq m$, let $r_i(t)$ be the rate at port i at time t. Then $r_i(t) = \alpha \cdot r_i(t-1) + (1-\alpha) \cdot a_i(t)$, where $a_i(t)$ is the number of packets that have just arrived at port i, and $\alpha \in (0, 1)$ is some fixed constant. We set $r_i(0) = 0$ initially. The overflow time for each port i is calculated as $t_i^{\mathrm{ov}}(t) = (B - b_i(t))/r_i(t)$, where $b_i(t)$ is the number of packets currently stored in buffer i. Note that $r_i(t)$ and $t_i^{\mathrm{ov}}(t)$ are allowed to take fractional values. At any time the algorithm serves the buffer that has the smallest overflow time; ties may be broken arbitrarily.

HSFOD is exactly 2-competitive [4] but, as mentioned above, has an excellent performance in practice. The algorithm depends on a parameter $0 < \alpha < 1$ that weights past and current packet arrivals. Experimental evaluations show that *HSFOD* achieves the best practical performance for values of α in the range $0.995 \leq \alpha \leq 0.997$.

We next turn to randomized online algorithms. Again, prior to our work Azar and Richter [7] had presented a *Random Schedule* algorithm that is $(\frac{e}{e-1})$-competitive and hence, in terms of competitiveness, never worse than deterministic strategies. Here we evaluate randomized algorithms against oblivious adversaries [11]. We have developed a randomized algorithm *Random Permutation* with a better performance. The basic approach of *Random Permutation* is to reduce the packet switching problem with m buffers of size B to one with mB buffers of size 1. To this end, a packet buffer q_i of size B is associated with a set $Q_i = \{q_{i,0}, \ldots, q_{i,B-1}\}$ of B buffers of size 1. A packet arrival sequence σ for the problem with size B buffers is transformed into a sequence $\tilde{\sigma}$ for unit-size buffers by applying a round robin strategy. More specifically, the j-th packet ever arriving at q_i is mapped to $q_{i,j \bmod B}$ in Q_i. *Random Permutation* at any time runs a simulation of the following algorithm *SimRP* for $m' = mB$ buffers of size 1. Basically, *SimRP* chooses a fixed, random permutation on the m' buffers and then, at any time, always serves the first non-empty buffer it encounters.

> **Algorithm SimRP(m').** The algorithm is specified for m' buffers of size 1. Initially, choose a permutation π uniformly at random from the permutations on $\{1, \ldots, m'\}$. In each step transmit the packet from the non-empty queue whose index occurs first in π.

The algorithm for buffers of arbitrary size then works as follows.

> **Algorithm Random Permutation.** Given a packet arrival sequence σ that arrives online, run a simulation of *SimRP(mB)* on $\tilde{\sigma}$. At any time, if *SimRP(mB)* serves a buffer from Q_i, transmit a packet from q_i. If the buffers of *SimRP(mB)* are all empty, transmit a packet from an arbitrary non-empty queue if there is one.

We proved that the algorithm achieves a nearly optimal competitive ratio.

Theorem 5. [21] *Random Permutation is 1.5-competitive.*

Theorem 6. [5] *The competitive ratio of any randomized online algorithm is not smaller than 1.4659, for any buffer size B.*

Further improvements are possible using resource augmentation or for architectures with small port numbers m. As for resource augmentation we have investigated two settings where an online algorithm (1) is granted additional buffer or (2) may serve at higher transmission rates. First suppose that an online strategy has an additional buffer $B' = cB$ per queue, where $c \geq 1$. We proved in this case that every work conserving algorithm is $(\frac{c+2}{c+1})$-competitive and that this bound is tight for the family of *Greedy* algorithms [5]. Next assume that an online algorithm may transmit $k \geq 1$ packets per time step, whereas an optimal strategy transfers one packet, as usual. In this scenario every work conserving online algorithm is $(1+\frac{1}{k})$-competitive [5]. As for small small port numbers, we showed that if $m = 2$, *Greedy* achieves a competitive ratio of $9/7 \approx 1.28$ while the competitiveness of deterministic strategies is not smaller than $16/12 \approx 1.2308$, see [21].

2.2 Experimental Study

In this section we report on the main results of an extensive experimental study [4] we have performed for the multi-buffer problem under consideration. In our study we have implemented all the algorithms mentioned in the previous section, i.e. *Greedy*, *Semi Greedy*, *Waterlevel*, *HSFOD*, *Random Schedule* and *Random Permutation*. Additionally, to determine competitive performance we have also implemented the optimal offline algorithm *SFOD*. Recall that *HSFOD* depends on a parameter α. We set α to 0.997 in the experiments as it yields the best results.

The main purpose of our experiments is to determine the experimentally observed competitiveness of the available online algorithms and to establish a relative performance ranking among the strategies. As the name suggests, the experimentally observed competitiveness is the ratio of the throughput of an online algorithm to that of an optimal solution as it shows in experimental tests. Additionally, we evaluated the running times and memory requirements of the algorithms as some of the strategies are quite involved and need auxiliary data structures. As for the running time of a strategy, we evaluated the *average time* it takes the algorithm to determine which queue to serve (total running time summed over all time steps/#time steps). With respect to extra space requirements, we have evaluated, for any of the algorithms, the maximum amount of memory needed by auxiliary data structures employed by that algorithm. Finally in our tests, we have evaluated the actual throughput in terms of the number of data packets transferred.

Setup. In order to get realistic and meaningful results, we have tested the online packet buffering algorithms on real-world traces from the Internet Traffic Archive [18], a moderated trace repository maintained by ACM SIGCOMM. We have performed extensive tests with seven traces. A first set of four traces monitors wide-area traffic between Digital Equipment Corporation (DEC) and the rest of the world. A second set of three traces monitors wide-area traffic between the Lawrence Berkeley Laboratory (LBL) and the rest of the world. Only the TCP traffic was considered. The traces were gathered over a time horizon of one to two hours and consist of 1.3 to 3.8 million data packets each. In the various traces the information relevant to us is, for any data packet, the arrival time and the sending host address.

In our experiments we have studied varying port numbers m as well as varying buffers sizes B. In order to be able to investigate varying values of m, we have to map sending host addresses to port numbers in the range $\{0, \ldots, m-1\}$. We chose a mapping that maps each sending host address to a port number chosen uniformly at random from $\{0, \ldots, m-1\}$. We would like to point out that such a mapping does not lead to balanced traffic at the ports as some hosts generate a large number of packets. In our traces, under the random mapping, we observe highly non-uniform packet arrival patterns where 10 to 15% of the ports receive ten times as many packets as each of the other ports. This is consistent with the fact that web traffic with respect to packets' source (and destination) addresses is distributed non-uniformly, exhibiting essentially a power-law structure [12,23]. Typically, 10% of the hosts account for 90% of the traffic.

Another important parameter in the experimental tests is the speed of the switch, i.e. how fast the switch can transfer packets. Here we consider speed values relative to the data volume of a given trace. For a trace data set D, let $f_D = (\#\text{packets in } D)/(\text{length of time horizon of } D)$ be the average packet arrival rate in D. Speed s indicates that the switch forwards data packets with frequency sf_D. Thus, intuitively, a speed-1 switch can forward the data exactly as fast as it arrives on the average. If the speed is low, the packet transmission rate is lower than the average arrival rate and, inevitably, buffers tend to be highly populated. On the other hand, if the speed is high, the transmission rate is higher than the arrival rate and buffers tend to be lightly populated.

Results. A first, very positive finding of our experiments is that the results are consistent for all the traces. In this survey we only present the plots for a trace named DEC-PKT-1 [18]. The phenomena reported in the following have occurred for all the data sets. In this survey we report on the competitiveness, running time and memory requirements of the algorithms as parameters m, B and s vary. It turns out that a variation of the speed s gives the most interesting results and we focus on them first.

Figure 2 depicts the experimentally observed competitiveness for varying s. In this survey we consider a basic setting with $m = 30$ and $B = 100$. These parameters are chosen relative to the size of DEC-PKT-1, which consists of 2.1 million packets. More precisely, we wish to simulate the algorithms for sufficiently large m and time steps with considerable packet traffic. This basic setting of m and B is not critical as the observed phenomena occur for other parameter settings (smaller/larger m and smaller/larger B) as well.

A first, interesting observation is that the performance of the three deterministic online algorithms *Greedy*, *Semi Greedy* and *Waterlevel* is almost identical in the experiments. Their curves are indistinguishable in Figure 2. For instance, the difference in the number of transferred packets is less than 1000 when the total throughput of each of the three strategies is about 2 million packets. All the three strategies have an experimental competitiveness that is always below 1.002, i.e. they are never 0.2% worse than an optimal solution. *HSFOD* exhibits an even better competitiveness of less than 1.001 for all values of s. In other words, *HSFOD* closes half of the gap between the other deterministic algorithms and the optimal offline strategy. In contrast, the randomized algorithms perform considerably worse than the deterministic algorithms, despite their better theoretical performance. Thus the theoretical and experimentally observed competitive ratios are unrelated.

Another important result of our study is that the experimentally observed competitiveness of all the algorithms ranges between 1.0 and 1.035 and hence is considerably lower than the theoretical bounds. This is not surprising because competitive analysis is a strong worst-case performance measure. It is astonishing, though, that the gap is so high. All the algorithms have the highest ratios for values of s around 1. Thus, the worst case occurs when the average packet

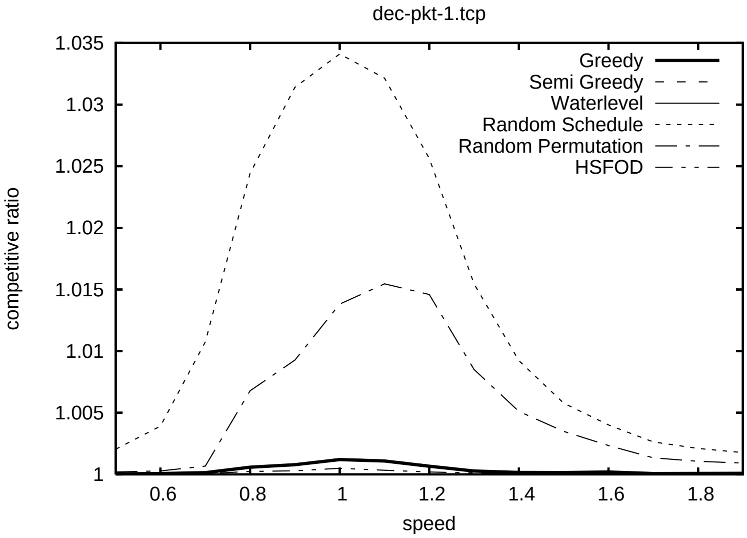


Fig. 2. Competitive ratio, $m = 30$ and $B = 100$

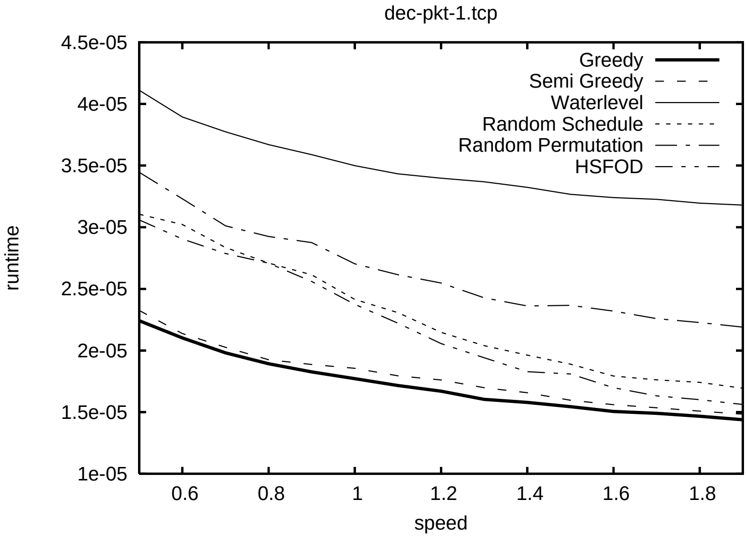


Fig. 3. Run time, $m = 30$ and $B = 100$

arrival rate is equal to the rate with which the switch can forward packets and packet scheduling decisions matter. For small and large values of s, the experimental competitiveness tends to 1. This is due to the fact that buffers tend to be either heavily populated (small s) or lightly populated (large s) and all the algorithms transfer essentially an optimum number of packets.

Figure 3 shows the running times of the algorithms, i.e. the average time in seconds it takes an algorithm to perform one time step (update auxiliary data structures to account for incoming packets and determine the queue to be served). We evaluate the running times for varying s because the buffer occupancy depends on s and the latter occupancy can affect the running time. Uniformly over all algorithms we observe decreasing running times for increasing values of s. The reason is that, for large s, buffers tend to be empty and the algorithms need less time to handle one time step. *Greedy* and *Semi Greedy* are the fastest algorithms, *Semi-Greedy* being only slightly slower than *Greedy*. *HSFOD*, *Random Schedule*, *Random Permutation* and *Waterlevel* have considerably higher running times. *Waterlevel* is the slowest strategy with running times that are more than twice as high as that of *Greedy*. A shown in Figure 3, the algorithms need 20 to 40 milliseconds to perform one time step. These times would be lower in a switch hardware implementation; our runtime tests just represent a comparative study of the algorithms.

As for the extra memory requirements of the algorithms, the numbers are stable for varying s. The important finding here is that the amounts of extra space differ vastly among the strategies. As to be expected, *HSFOD*, *Greedy* and *Semi Greedy* have small requirements. *HSFOD* uses no more than 500 bytes, while *Greedy* and *Semi Greedy*, using priority queues, allocate 1000 to 1300 bytes. *Waterlevel* has space requirements that are twice as high. Huge amounts of extra space (80.000 to 100.000 bytes) are required by *Random Schedule* and *Random Permutation* because these algorithms need space for auxiliary queues and mB unit-size buffers.

We briefly summarize the results for varying buffer size B and for a varying number m of ports. Here the experiments were done for the critical speed $s = 1$. As B varies, the experimentally observed competitive ratios of the algorithms are stable. The running times are stable, too. The only exception is *Random Schedule* exhibiting slightly increasing running times as B increases. This is due to the fact that the algorithm has to maintain auxiliary queues whose size depends on B. As for the required space, all the deterministic strategies have fixed demands as the size of the auxiliary data structures depends only on m. *Random Schedule* and *Random Permutation* experience a linear increase as the auxiliary data structures depend on mB.

As for varying port number m, all the algorithms show a (very) slight increase in experimental competitiveness. The increase is more pronounced in the case of the randomized algorithms. The general increase in competitiveness is based on the fact that for a larger number of ports, online algorithms have a higher chance of serving the "wrong" port. Analyzing running time we observe a weakness of *HSFOD*; its running time increases linearly with m. The same holds for *Waterlevel*, although the gradient is smaller here. For all the other strategies the running times are stable.

In summary, the main conclusion of our experimental study is that greedy-like strategies and *HSFOD*, despite their higher theoretical competitiveness, are the methods of choice in a practical environment.

3 Web Caching

Web caching is a fundamental resource management problem that arises when users, working on computers, surf the web and download documents from other network sites. Downloaded documents can be stored in local caches so that they do not have to be retransmitted when users wish to access these documents again. Caches can be maintained by web clients or servers. Storing frequently accessed documents in local caches can substantially reduce user response times as well as the network congestion. Web caching differs from standard paging and caching in operating systems in that documents here have varying sizes and incur varying costs when being downloaded. The loading cost depends, for instance, on the size of the document and on the current congestion in the network. The goal is to maintain the local caches in such a way that a sequence of document accesses generated in the network can be served with low total loading cost. It turns out that this distributed problem can be decomposed into local ones, one for each cache in the network: Note that the cache configurations are essentially independent of each other. Each cache may store an arbitrary subset of the documents available in the network. Thus the problem of minimizing the loading cost globally in the network is equivalent to minimizing the loading cost incurred locally at each cache. Therefore, the following problem abstraction has been studied extensively.

We are given a two-level memory system consisting of a small fast memory and a large slow memory. The fast memory represents a local cache we have to maintain, which may reside in a web server or browser. The slow memory represents the remaining network, i.e. the universe of all documents accessible in the network. We assume that the fast memory has a capacity of K, counted in bits. For any document d, let $\mathit{size}(d)$ be the size and $\mathit{cost}(d)$ be the cost of d. The size is again measured in bits. A sequence of requests for documents arrives at the fast memory. At any time one request can be served. If the referenced document is in cache, the request can be served with 0 cost. If the requested document is not in cache, it must be retrieved over the network at a cost of $\mathit{cost}(d)$. Immediately after this download operation the document may be brought at no extra cost into cache. We emphasize here that such a loading operation is optional. In web applications referenced documents are not necessarily brought into cache. This technique is referred to as *bypassing*. The goal is to serve a sequence of requests so that the total loading cost is as small as possible. Various cost models have been proposed in the literature.

1. *The Bit Model*: For each document d, we have $\mathit{cost}(d) = \mathit{size}(d)$. (The delay in bringing the document into fast memory depends only upon its size.)
2. *The Fault Model*: For each document d, we have $\mathit{cost}(d) = 1$ while the sizes can be arbitrary.
3. *The General Model*: For each document d, both the cost and size can be arbitrary.

In the following, let $k = K/D_{\min}$ be the ratio of the cache size K to the size $D_{\min}$ of the smallest document ever requested.

For the Bit and the Fault Model, Irani [17] gave randomized online algorithms that are $O(\log^2 k)$-competitive. She also gave polynomial time offline strategies that achieve approximation ratios of $O(\log k)$. We remark that the offline problem is NP-hard in the Bit and hence in the General Model. This follows from a simple reduction from the Partition Problem. Interestingly, the complexity is unknown for the Fault Model. For the General Model Young [24] presented a deterministic online algorithm called *Landlord* that is k-competitive. He assumes that bypassing is not allowed, i.e. a requested document must be brought into cache in order to serve a request. Recently, Bansal et al. [8] devised a randomized $O(\log^2 k)$-competitive online algorithm for the General Model. They also gave randomized strategies achieving an improved competitiveness of $O(\log k)$ in the Bit and Fault Models.

The above web caching problem assumes that requests must be served in the order of arrival. However, this assumption is often too restrictive. Current versions of HTTP support pipelining, where multiple requests may arrive simultaneously at a web server or a proxy. Moreover, if a proxy receives web requests from different data streams, these references do not have to be served in the strict order of arrival. In such settings request reordering is a promising approach to improve cache hit rates. Therefore, Feder et al. [13,14] recently initiated the study of web caching assuming that a limited reordering of requests is allowed. Formally in *web caching with request reordering*, requests do not have to be served in the order of arrival. Of course, it is not desirable to delay the service of a request for too long. Let r be a positive integer. Request $\sigma(j)$ may be served before $\sigma(i)$ if $j - i < r$. We also refer to this setting as the *r-reordering* problem.

The r-reordering problem has also been studied for the Uniform Model which is motivated by paging problems arising in operating systems.

1. *Uniform Model*: All documents have the same size and incur a cost of 1 when not available in cache, i.e. $size(d) = s$ and $cost(d) = 1$, for some positive integer s and all documents d.

In a first paper, Feder et al. [13] studied the case that at any time the cache can store only one document, i.e. the cache size is 1. For all cost models, they gave constant competitive online algorithms. Furthermore they developed an offline algorithm based on dynamic programming that achieves a polynomial running time if r is logarithmic in the length of the request sequence or if the number of distinct documents requested is constant. In an extended version [14] of their paper, Feder et al. also presented online algorithms for arbitrary cache sizes. Suppose that a cache can simultaneously store $k' = \lfloor k \rfloor$ documents. For the Uniform Model, Feder et al. [14] gave deterministic $(k' + 2)$-competitive algorithms. For the Bit and Fault Models, they showed $(k' + 3)$-competitive strategies. All of these results assume that bypassing is not allowed.

In the following two sections we review our new contributions [1] on web caching with request reordering. We first study online algorithms and then address offline approximations.

Algorithm Modified Landlord (MLL)

```
Let $W$ be a window of $r$ consecutive requests, the first of which may be unserved.
For all $d' \in$ cache such that $W$ contains unserved requests to $d'$,
serve those requests;
if first request in $W$ is unserved then
    Let $d$ be the document referenced by this first request;
    Serve all requests to $d$ in $W$;
    Set $credit(d) \leftarrow cost(d)$ and $S \leftarrow \{d\} \cup \{d' \mid d' \in \text{cache}\}$;
    while $\sum_{d' \in S} size(d') > K$ do
        Let $\Delta = \min_{d' \in S} credit(d')/size(d')$;
        For each $d' \in S$, decrease $credit(d')$ by $\Delta size(d')$;
        Delete from $S$ and the cache any document $d'$ with $credit(d') = 0$;
    if $credit(d) > 0$ then bring $d$ into cache;
Shift $W$ one position to the right;
```

Fig. 4. The Modified Landlord algorithm

3.1 An Optimal Online Algorithm

We have presented a deterministic online algorithm for the r-reordering problem that achieves an optimal competitive ratio in the General Model. The algorithm is a modification of Young's *Landlord* algorithm [24]. Any document d has a credit that takes values between 0 and $cost(d)$. Initially, all documents have a credit of 0. Given a request sequence σ, the algorithm *Modified Landlord*, depicted in Figure 4, is executed. The algorithm maintains a sliding window W that always contains r consecutive requests of σ. Requests to the left of W are served and the first request in W may constitute an unserved request. In a first step *Modified Landlord* serves all requests in W that are made to documents residing in cache. These service operations do not incur cost. If the first request in W, say to a document d, is indeed unserved, the algorithm serves the references to d in W at a cost and then determines whether or not to bring d into cache. To this end the algorithm sets the credit of d to $cost(d)$ and considers the set S formed by document d and the documents in cache. While the total size of S exceeds the cache capacity the algorithm reduces credit values according to the standard *Landlord* algorithm, i.e. by considering smallest relative credit values. Documents with a credit of 0 are discarded from S. After these operations, W is shifted by one position to the right.

Recall that $k = K/D_{\min}$, where $D_{\min}$ is the size of the smallest document that can be referenced.

Theorem 7. [1] *The algorithm MLL is* $(k+1)$*-competitive.*

The competitiveness of $k+1$ is best possible for deterministic online algorithms that allow bypassing, see [15,18].

3.2 Offline Approximation Algorithms

We have developed polynomial time offline algorithms that achieve constant factor approximation ratios. Our algorithms are based on a general technique that transforms the r-reordering problem into one of computing batched service schedules. We first present this technique and then demonstrate that it can be used to develop approximation algorithms for the various cost models.

As in the online setting we imagine that an algorithm, processing a request sequence, maintains a sliding window W that always contains r consecutive requests. Requests to the left of W are served and request reordering is feasible within the window. We say that an algorithm A *serves a request sequence in batches* if, for any $i = 0, 1, \ldots, \lceil m/r \rceil - 1$, A serves all requests $\sigma(ir+1), \sigma(ir+2), \ldots, \sigma(\min\{ir+r, m\})$ when $\sigma(ir+1)$ is the leftmost request in the sliding window W. Requests $\sigma(ir+1), \ldots, \sigma(\min\{ir+r, m\})$ are also referred to as *batch i*. Thus, when $\sigma(ir+1)$ becomes the leftmost request in W, all requests in batch i are still unserved. The batch is served while the position of W remains unchanged. Then W is shifted to $\sigma((i+1)r+1)$ if $i < \lceil m/r \rceil - 1$. For any batch i, let $B(i)$ be the set of documents referenced in that batch. The following lemma implies that one only loses a factor of 2 in terms of performance when considering batched service schedules.

Lemma 1. [1] *Let A be an algorithm that serves a request sequence σ at cost C in the standard r-reordering model. Then there exists an algorithm A' that serves σ in batches and incurs a cost of at most $2C$.*

The Uniform Model. Recall that in this basic setting all documents have the same size and incur a cost of 1 when being served or loaded into cache. We have devised a batched version of Belady's [10] optimum offline paging algorithm MIN, taking into account that requested documents do not necessarily have to be brought into cache. On a cache replacement, the algorithm evicts a document whose next unserved request occurs in the highest indexed batch possible. We use the following notation. Consider an algorithm that serves a request sequence in batches. At any given time during the processing and for any document d, let $b(d)$ be the index of the batch where the next unserved request to d occurs. If d is not requested again, let $b(d) = \lceil m/r \rceil$.

Algorithm BMIN. Serve a request sequence in batches. When the processing of a batch starts, first serve all requests to documents that are currently in cache. While there is still a document d with unserved requests in the batch, execute the following steps. Serve all requests to d and determine $b = \max_{d' \in S} b(d')$, where S is the set of documents that are currently in cache. If $b(d) < b$, load d into cache and evict any document d' with $b(d') = b$.

Theorem 8. [1] *For any request sequence σ, BMIN incurs the minimum cost among algorithms processing request sequences in batches.*

Using Lemma 1 we obtain the following result.

Corollary 1. [1] *BMIN achieves an approximation ratio of 2.*

The Bit Model. In the Bit Model *cost*(d) = *size*(d), for any document d. Again we design an approximation algorithm that processes a request sequence in batches. The algorithm proceeds in two steps. First it constructs a *fractional solution*, where documents are allowed to be fractionally in cache. We say that a document d *is fractionally in cache* if $0 < c(d) < \mathit{size}(d)$, where $c(d)$ denotes the number of bits of d present in cache. In this case the cost of serving requests to d or loading the remainder of d into cache is equal to $\mathit{cost}(d) - c(d)$. In a second step the algorithm rounds the fractional solution to a feasible integral solution. The strategy for designing fractional solutions is a bitwise implementation of BMIN. For any document d and any batch i, let $b(d,i)$ be the smallest index j, $j > i$, such that batch j contains requests to d. If d is not requested again after batch i, then $b(d,i) = \lceil m/r \rceil$.

Algorithm BBMIN. Serve the request sequence in batches. For any batch i, first serve the requests to documents that are fully or fractionally in cache; consider those documents d in non-increasing order of $b(d,i)$ values. Then serve the requests to documents not present in cache. For any requested document d that is not fully in cache, execute the following steps after the service operation. Determine $b = \max_{d' \in S} b(d',i)$, where S is the set of documents that are fully or fractionally in cache. While $b(d,i) < b$ and $c(d) < \mathit{size}(d)$, perform two instructions: (1) Evict $\beta = \min\{\mathit{size}(d) - c(d), c(d')\}$ bits from any document d' with $b(d',i) = b$. (2) Load β missing bits of d and recompute $b = \max_{d' \in S} b(d',i)$.

Next we present our rounding algorithm, generating an integral solution from the fractional solution produced by BBMIN. We extend the notation. For any document d and any batch i, let $c(d,i)$ be the number of bits of d present in cache when the processing of batch i ends. We first modify the solution of BBMIN such that the extent to which a document is in cache does not change between two consecutive batches in which the document is requested. Suppose that d is referenced in batch i. Document d's presence in cache may decrease during the processing of batches $j = i+1, \ldots, b(d,i)-1$. The cost of the next reference to d is equal to the number of bits of d not present in cache when batch $b(d,i)-1$ ends, which is equal to $\mathit{size}(d) - c(d, b(d,i)-1)$. Therefore, for any document d and batch i such that $d \in B(i)$, we set $c(d,j) = c(d, b(d,i)-1)$, for $j = i, \ldots, b(d,i)-2$. This does not change the cost of the solution and only frees up space in cache. The solution by BBMIN has an important property that is crucial for the actual rounding procedure: Consider two batches i and j with $i \le j$ as well as two documents $d \in B(i)$ and $d' \in B(j)$. If $b(d',j) < b(d,i)$ and $c(d,l) > 0$ for $l = i, \ldots, b(d,i)-1$, then $c(d',l) = \mathit{size}(d')$, for $l = j, \ldots, b(d',j)-1$. This is because BBMIN prefers d over d' when evicting bits as d's next request is farther in the future.

The rounding procedure produces a solution that may need some extra space in cache. More precisely, the solution may need up to $K + \delta D_{\max}$ memory in cache, where $D_{\max}$ is the size of the largest document ever requested and δ is an arbitrary constant satisfying $0 < \delta \le 1$. Later, in order to achieve the desired approximation guarantee, we will set $\delta = 1/(1+\epsilon)$. We remark that in practical applications the extra space requirement is small; typically $D_{\max}$ comprises not more than 1-2% of the cache size. Fix a δ with $0 < \delta \le 1$. For batches $i = 0, \ldots, \lceil m/r \rceil - 1$ the procedure considers the documents $d \in B(i)$ with $(1-\delta)\mathit{size}(d) < c(d,i) < \mathit{size}(d)$. Document d is rounded up if, after the rounding, the rounded-up documents occupy extra memory of no more than $\delta D_{\max}$. Otherwise the document is rounded down to $(1-\delta)\mathit{size}(d)$. Finally, all values $c(d,i)$ with $0 < c(d,i) \leq (1-\delta)\mathit{size}(d)$ are rounded down to 0. The pseudo-code is given in Figure 5 below.

A thorough analysis of the above scheme, using $\delta = 1/(1+\epsilon)$, yields the following result.

Theorem 9. [1] *For any $\epsilon \geq 0$, we can construct a solution that incurs a cost of at most $(1+\epsilon)\mathrm{BBMIN}(\sigma)$ and uses an additional memory of at most $D_{\max}/(1+\epsilon)$.*

Again, using Lemma 1 we obtain:

Corollary 2. [1] *For any request sequence σ and any $\epsilon \geq 0$, we can construct a solution that incurs a cost of at most $2+\epsilon$ times that of an optimal solution. The extra space required is bounded by $D_{\max}/(1+\epsilon/2)$.*

The Fault Model. We investigate the Fault Model where $\mathit{cost}(d) = 1$, for all documents d, and design an approximation algorithm that processes a given request sequence in batches. As in the previous section, for any document d and any batch i, let $b(d,i)$ be the smallest index $j > i$ such that batch j contains requests to d and let $c(d,i)$ be the number of bits of documents d present in cache when the processing of batch i ends. The number of bits of d that are in cache initially is denoted by $c(d,-1)$. Unfortunately, we know of no simple combinatorial algorithm for constructing a good fractional solution operating in batches. Therefore, we formulate the caching problem as a linear program.

W.l.o.g. we may restrict ourselves to solutions in which the extent to which a document d is in cache does not change between two consecutive batches referencing d. Thus if $d \in B(i)$, then $c(d,j) = c(d, b(d,i)-1)$, for $j = i, \ldots, b(d,i)-2$. The cost for serving requests to $d \in B(i)$ is equal to the fraction to which d is not in cache at the end of the previous batch, which is $1 - c(d,i-1)/\mathit{size}(d)$. The

Algorithm Rounding

```
$\mathit{Extra} \leftarrow 0$;
for $i \leftarrow 0$ to $\lceil m/r \rceil - 1$ do
    for all $d \in B(i)$ with $(1-\delta)\mathit{size}(d) < c(d,i) < \mathit{size}(d)$ do
        if $\mathit{Extra} + \mathit{size}(d) - c(d,i) \leq \delta D_{\max}$ then
            $\mathit{Extra} \leftarrow \mathit{Extra} + \mathit{size}(d) - c(d,i)$;
            $c(d,i) \leftarrow \mathit{size}(d)$,  for $j = i, \ldots, b(d,i) - 1$;
        else
            $\mathit{Extra} \leftarrow \mathit{Extra} - (c(d,i) - (1-\delta)\mathit{size}(d))$;
            $c(d,i) \leftarrow (1-\delta)\mathit{size}(d)$,  for $j = i, \ldots, b(d,i) - 1$;
for all $c(d,i)$ with $0 < c(d,i) \leq (1-\delta)\mathit{size}(d)$ do
    $c(d,i) \leftarrow 0$;
```

Fig. 5. The Rounding algorithm

only additional constraint we have to impose is that the total size of documents in cache may not exceed K. Thus the linear program is as follows. Let D be the set of documents ever requested.

$$\text{Minimize} \quad \sum_{i=0}^{\lceil m/r\rceil-1} \sum_{d\in B(i)} (1 - c(d, i-1)/\mathit{size}(d))$$

subject to

$$\begin{array}{ll}
c(d,j) = c(d, b(d,i) - 1)) & \forall d, i, j \text{ such that} \\
 & d \in B(i) \text{ and} \\
 & i \le j < b(d,i) - 1 \\
\sum_{d\in D} c(d,i) \le K & \forall i \\
c(d,i) \in \{0, \mathit{size}(d)\} & \forall i, d
\end{array}$$

We replace the constraint $c(d,i) \in \{0, \mathit{size}(d)\}$ by $0 \le c(d,i) \le \mathit{size}(d)$ so that the linear programming relaxation can be solved in polynomial time. Let $\mathrm{OPT}^B_{\mathrm{LP}}(\sigma)$ be the cost incurred by the LP relaxation on input σ. Clearly, this is a lower bound on the minimum cost of any integral solution processing σ in batches.

We have to round the optimal fractional solution to obtain a feasible integral solution. To this end we fix an $\epsilon > 0$ and partition the documents into classes such that, within each class, the size of the documents differ by a factor of at most $1+\epsilon$. Formally, we partition the documents into $\lfloor \log_{1+\epsilon} D_{\max}\rfloor + 1$ classes such that class C_k, $0 \le k \le \lfloor \log_{1+\epsilon} D_{\max}\rfloor$ contains documents d with $D_{\max}(1+\epsilon)^{-k} \ge \mathit{size}(d) > D_{\max}(1+\epsilon)^{-(k+1)}$. We assume that each document consists of at least 1 bit. In the following we will round the optimal LP relaxation solution and essentially apply the Rounding algorithm given in Figure 5 separately to each class. In order to be able to do so, we first modify the optimal LP relaxation solution such that, for any class, documents requested earlier in the future reside to a larger extent in cache. Formally, we require that, for each class C_k, the following property holds. Let $d, d' \in C_k$ be two documents with $d \in B(i)$, $d' \in B(j)$ and $i \le j$. If $b(d',j) < b(d,i)$ and $c(d,l) > 0$ for $l = i, \ldots, b(i,d) - 1$ then $c(d',l) = \mathit{size}(d')$, for $l = j, \ldots, b(d',j) - 1$. This property corresponds to that in the Bit Model, which was essential to apply the Rounding algorithm. The property can be established relatively easily by updating cache extents properly whenever there is a conflict for two documents i and j; details can be found in [1]. Let OPT^* be the solution obtained after all these modifications.

Given OPT^*, we apply the Rounding algorithm described in Figure 5 separately for each class C_k. We use parameter $\delta = 1$ and replace $D_{\max}$ by $D^k_{\max}$, where $D^k_{\max}$ is the size of the largest document in C_k. An analysis of the entire scheme leads to the following result.

Theorem 10. [1] *For any request sequence σ and for any $\epsilon > 0$, we can construct a solution that processes σ in batches and incurs a cost of at most $(1+\epsilon)\mathrm{OPT}^B_{\mathrm{LP}}(\sigma)$. The additional memory used by the solution is at most $D_{\max}(1 + 1/\epsilon)$.*

Corollary 3. [1] *For any request sequence σ and any $\epsilon > 0$ we can construct a solution that incurs a cost of at most $2 + \epsilon$ times the optimum cost and uses an extra space of no more than $(1 + 2/\epsilon)D_{\max}$.*

The General Model. In the General Model the service/loading cost $cost(d)$ can be an arbitrary value, for any document d. Our solution for the General Model is based on an approximation algorithm by Bar-Noy et al. [9]. They considered the following *loss minimization problem*. Let $\mathcal{I}$ be a set of intervals $I = [t_1, t_2]$, the scheduling of which requires a resource of bounded availability. Let $width(t)$ be the amount of resource available at time t. Each interval $I \in \mathcal{I}$ has a width $w(I)$ as well as a penalty $p(I)$. The width reflects the amount of resource required at any time $t \in I$ if I is scheduled. The penalty represents the cost if I is not scheduled. The goal is to find a set $\mathcal{S} \in \mathcal{I}$ of intervals being scheduled such that $\sum_{I \in \mathcal{S}} w(I) \leq width(t)$, for any time t, and the total penalty $\sum_{I \in \mathcal{I} \setminus \mathcal{S}} p(I)$ is as small as possible. Bar-Noy et al. showed that the standard web caching problem, where request reordering is not allowed and documents must be in cache at the time of the reference, can be formulated as a loss minimization problem. We showed [1] that the problem of constructing a schedule that processes a given request sequence in batches and allows bypassing can also be formulated in the above framework.

Theorem 11. [1] *For any σ, we can construct a solution that processes σ in batches and incurs a cost of at most 4 times that of an optimal solution that serves σ in batches. No extra space is required.*

Corollary 4. [1] *For any σ, we can construct a solution that incurs a cost of at most 8 times that of an optimal solution. No extra space is required.*

4 Conclusions

This survey has reviewed algorithmic results for two basic memory management problems in large network. While, for both of the problems, a comprehensive set of results has been developed, several interesting open questions remain. For the multi-buffer management problem in network switches, the presented algorithms do not take into account fairness constraints providing a certain Quality-of-Service to each data stream. Moreover, packet deadlines are ignored. In practice, if a packet does not reach its destination by a certain point in time, it is considered lost. It would be interesting to integrate both aspects into the investigations. Moreover, for the algorithms investigated, there is a huge gap between the theoretical and experimentally observed performance. A challenging problem is to characterize packet sequences arising in practice and to develop refined analyses of the available strategies such as *Greedy*. As for web caching with request reordering, it would be interesting to extend the online algorithm to a scenario without bypassing. This extension turns out to be relatively straightforward for our offline strategies. Nonetheless, for the offline setting, it would be good to develop algorithms with even improved approximation guarantees.

References

1. Albers, S.: New results on web caching with request reordering. In: Proc. 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 84–92 (2004)
2. Albers, S.: On the value of coordination in network design. In: Proc. 19th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 294–303 (2008)
3. Albers, S., Eilts, S., Even-Dar, E., Mansour, Y., Roditty, L.: On Nash equilibria for a network creation game. In: Proc. 17th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 89–98 (2006)
4. Albers, S., Jacobs, T.: An experimental study of new and known online packet buffering algorithms. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 754–765. Springer, Heidelberg (2007)
5. Albers, S., Schmidt, M.: On the performance of greedy algorithms in packet buffering. In: Proc. 36th ACM Symposium on Theory of Computing (STOC), pp. 35–44 (2004); Extended version in SIAM Journal on Computing 35(2), 278–304 (2005)
6. Azar, Y., Litichevskey, A.: Maximizing throughput in multi-queue switches. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 53–64. Springer, Heidelberg (2004)
7. Azar, Y., Richter, Y.: Management of multi-queue switches in QoS networks. Algorithmica 43, 81–96 (2005)
8. Bansal, N., Buchbinder, N., Naor, J.: Randomized competitive algorithms for generalized caching. In: Proc. 40th Annual ACM Symposium on Theory of Computing (STOC), pp. 235–244 (2008)
9. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. Journal of the ACM 48, 1069–1090 (2001)
10. Belady, L.A.: A study of replacement algorithms for virtual storage computers. IBM Systems Journal 5, 78–101 (1966)
11. Ben-David, S., Borodin, A., Karp, R.M., Tardos, G., Wigderson, A.: On the power of randomization in on-line algorithms. Algorithmica 11, 2–14 (1994)
12. Elhanany, I., Chiou, D., Tabatabaee, V., Noro, R., Poursepanj, A.: The network processing forum switch fabric benchmark specifications: An overview. IEEE Network, 5–9 (2005)
13. Feder, T., Motwani, R., Panigrahy, R., Zhu, A.: Web caching with request reordering. In: Proc. 13th ACM-SIAM Symposium on Discrete Algorithms, pp. 104–105 (2002)
14. Feder, T., Motwani, R., Panigrahy, R., Seiden, S., van Stee, R., Zhu, A.: Combining request scheduling with web caching. Extended version of [13]. Theoretical Computer Science 324, 201–218 (2004)
15. Feldmann, A., Karlin, A., Irani, S., Phillips, S.: Private communication, transmitted through [17] (1996)
16. Hoefer, M., Souza, A.: Tradeoffs and average-case equilibira in selfish routing. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 63–74. Springer, Heidelberg (2007)
17. Irani, S.: Page replacement with multi-size pages and applications to Web caching. Algorithmica 33, 384–409 (2002)
18. The Internet traffic archive, `http://ita.ee.lbl.gov`
19. Jacobs, T.: Constant factor approximations for the hotlink assignment problem. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 188–200. Springer, Heidelberg (2007)

20. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. In: Proc. 33rd Annual ACM Symposium on Theory of Computing, pp. 520–529 (2001)
21. Schmidt, M.: Packet buffering: Randomization beats deterministic algorithms. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 293–304. Springer, Heidelberg (2005)
22. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28, 202–208 (1985)
23. Williamson, C.: Internet traffic measurements. IEEE Internet Computing 5, 70–74 (2001)
24. Young, N.E.: Online file caching. Algorithmica 33, 371–383 (2002)

Multicast Routing and Design of Sparse Connectors

Andreas Baltz and Anand Srivastav

Institut für Informatik
Christian-Albrechts-Universität Kiel
Christian-Albrechts-Platz 4
24118 Kiel

Abstract. In a multicast communication network, each sender communicates with multiple nodes which are in request of identical data. We summarize our studies of the minimum multicast congestion problem, which generalizes the well-known NP-hard multicommodity flow problem. Moreover, we describe efficient architectures for a network allowing n senders to connect to N receivers ($n < N$).

1 Introduction

In the first phase of the program, we studied the *Minimum Multicast Congestion Problem*. Here, a communication network is modeled as an undirected graph $G = (V, E)$, where each node represents a computer that is able to receive, copy and send packets of data. *Multicast* traffic means that several nodes have a simultaneous demand to receive a copy of a single packet. These nodes together with the packet's source specify a *multicast request* $S \subseteq V$. Meeting the request means to establish a connection represented by a *Steiner tree* with *terminal set* S, i.e., a subtree of G that contains S and has no leaf in $V \setminus S$. Given G and a set of multicast requests it is natural to ask about Steiner trees such that the maximum *edge congestion*, i.e., the maximum number of trees sharing an edge is as small as possible. Solving this minimization problem is NP-hard, since it contains as a special case the well-known NP-hard standard routing problem of finding (unsplittable) paths with minimum congestion. The MAX-SNP-hard minimum Steiner tree problem is also closely related.

The second phase was concerned with the design of communication networks using few connections between n inputs and N outputs, and still allowing all the inputs to send information to arbitrary distinct outputs simultaneously. In the usual graph model, this network design problem can be stated as follows.

Given $n, N \in \mathbb{N}$ ($n \leq N$), construct a digraph $G = (V, E)$, where $V = I \cup L \cup O$ is partitioned into input vertices, intermediate vertices and output vertices such that

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 247–265, 2009.

- $|I| = n$, $|O| = N$,
- for every injective mapping $f : I \to O$ there are vertex disjoint paths connecting i to $f(i)$ for all $i \in I$,
- $|E|$ is small, or even minimum.

We call a digraph as above an (n, N)*-connector* (well-known in literature also as *rearrangeable network*, *permutation network*, and (N, n)*-permuter*). An (n, N, k)*-connector* is an (n, N)-connector, where each path from I to O is of length k.

The results of the second phase have been obtained in joint work with Gerold Jäger, and – in part – with Amnon Ta-Shma.

In the following we will survey our results and give brief sketches of some of the proofs. Comprehensive versions can be found in [1,2,3].

2 Minimum Multicast Congestion Problem

2.1 Previous Work

Vempala and Vöcking [4] gave a randomized algorithm for approximating the minimum multicast congestion problem within a factor of $O(\log n)$ by applying randomized rounding to an integer linear program relaxation. The considered LP contains an *exponential* number of constraints corresponding to the exponential number of possible Steiner trees. Vempala and Vöcking handled this difficulty by considering a multicommodity flow relaxation for which they could devise a polynomial separation oracle. By rounding the fractional paths in an $O(\log n)$-stage process they proved an $O(\log n)$ approximation. Carr and Vempala [5] gave an algorithm for approximating the minimum multicast congestion within a constant factor *plus* $O(\log n)$. They showed that an r-approximate solution to an *LP*-relaxation can be written as a convex combination of Steiner trees. Randomized rounding yields a solution not exceeding a congestion of $\max\{2\exp(1) \cdot r\mathrm{OPT}, 2(\varepsilon + 2)\log n\}$ with probability at least $1 - n^{-\varepsilon}$, where r is the approximation factor of the network Steiner problem. Both algorithms use the ellipsoid method with separation oracle and thus are of mere theoretical value. A closely related line of research is concerned with combinatorial approximation algorithms for multicommodity flow and – more general – fractional packing and covering problems. Matula and Shahrokhi [6] were the first to develop a combinatorial strongly polynomial approximation algorithm for the uniform concurrent flow problem. Their method was generalized and improved by Goldberg [7], Leighton et al. [8], Klein et. al. [9], Plotkin et al. [10], and Radzik [11]. A fast version that is particularly simple to analyze is due to Garg and Könemann [12]. Independently, similar results were given by Grigoriadis and Khachiyan [13]. Jansen and Zhang [14] extended the latter approach. In particular, they presented a randomized algorithm for approximating the minimum multicast congestion within $(1+\varepsilon)(r\mathrm{OPT}+\exp(1)\ln m)$ in time $O(m(\ln m+\varepsilon^{-2}\ln\varepsilon)(k\beta + m\ln\ln(m/\varepsilon))$, where β is the running time of a minimum

Steiner tree approximation. The online version of the problem was considered by Aspnes et al. [15].

2.2 Our Results

We present three algorithms which solve the *fractional* multicast congestion problem up to a relative error of $r(1+\varepsilon)$. The fastest of these takes time $O(k\beta\varepsilon^{-2}\ln k \ln m)$ (where β bounds the time for computing an r-approximate minimum Steiner tree). By means of randomized rounding we obtain a $(1+\varepsilon)(r\mathrm{OPT}+\exp(1)\ln m)$-approximation to the *integral* multicast congestion problem. With the tools of Srivastav and Stangier [16] the rounding procedure can be derandomized. The three algorithms are based on the approaches of Plotkin, Shmoys and Tardos [10], Radzik [11], and Garg and Könemann [12] to path-packing and general packing problems. In experiments it turned out that straightforward implementations of the above (theoretically fast) combinatorial algorithms are quite slow. However, simple modifications lead to a very fast algorithm with much better approximation results than hinted at by the worst case bounds.

2.3 Notations

Let $G=(V,E)$ denote the given undirected graph on n nodes and m edges. We consider k multicast requests, $S_1,\dots,S_k \subseteq V$. The set of all Steiner trees with terminal nodes given by S_i is denoted by $\mathcal{T}_i$. As to the (possibly fractional) congestion, we distinguish between $c_i(T)$, the congestion caused by tree T for request S_i,

$$c_i(e) := \sum_{\substack{T\in\mathcal{T}_i\\ e\in E(T)}} c_i(T),$$

the congestion of edge $e\in E$ due to request S_i, $c(e) := \sum_{i=1}^{k} c_i(e)$, the total congestion of edge $e\in E$, and $c_{\max} := \max_{e\in E} c(e)$, the maximum edge congestion. We will define a nonnegative length function l on the edges and abbreviate the sum of edge lengths of a subgraph U of G by $l(U) := \sum_{e\in E(U)} l(e)$. The time for computing an r-approximate optimum Steiner tree will be denoted by β. Throughout, $r \geq 1.55$ and $\varepsilon \in (0,1]$ will be constant parameters describing the guaranteed ratio of our minimum Steiner tree approximation [17] and determining the target approximation of the fractional optimization problem, respectively. $\mathrm{MST}_l(S_i)$ and $\widetilde{\mathrm{MST}}_l(S_i)$ are used to denote a minimum Steiner tree and an approximate minimum Steiner tree for S_i with respect to l (l is usually omitted); we assume $l(\widetilde{\mathrm{MST}}(S_i)) \leq r\cdot l(\mathrm{MST}(S_i))$. The optimal fractional congestion is denoted by OPT.

2.4 LP Formulation

The minimum multicast congestion problem can be formulated as an integer linear program. We study a natural *LP* relaxation and its dual:

$$\begin{array}{lll}
\text{(LP)} & \min z & \\
& \text{s. t.} & \\
& \sum_{T\in\mathcal{T}_i} c_i(T) \geq 1 & \text{for all } i \in \{1,\ldots,k\} \\
& c(e) = \sum_{i=1}^k \sum_{\substack{T\in\mathcal{T}_i \\ e\in E(T)}} c_i(T) \leq z & \text{for all } e \in E \\
& c_i(T) \geq 0 & \text{for all } i \text{ and all } T \in \mathcal{T}_i
\end{array}$$

$$\begin{array}{lll}
\text{(LP}^*\text{)} & \max \sum_{i=1}^k Y_i & \\
& \text{s. t.} & \\
& \sum_{e\in E} l(e) \leq 1 & \\
& \sum_{e\in E(T)} l(e) \geq Y_i & \text{for all } i \text{ and all } T \in \mathcal{T}_i \\
& l(e) \geq 0 & \text{for all } e \in E \\
& Y_i \geq 0 & \text{for all } i \in \{1,\ldots,k\}
\end{array}$$

We are going to use this formulation as the basis for combinatorial algorithms. Since these algorithms are proven to run in polynomial time, we will get solutions containing only polynomially many non-zero variables, despite the fact that the LP formulation has exponential size.

Lemma 1. *Let* $(Y_1^*,\ldots,Y_k^*,l^* : E \to \mathbb{R}_{\geq 0})$ *be a feasible solution to (LP***).* $(Y_1^*,\ldots,Y_k^*,l^* : E \to \mathbb{R}_{\geq 0})$ *is an optimal dual solution if and only if*

$$\sum_{i=1}^k Y_i^* = \sum_{i=1}^k l^*(MST(S_i)) = \max\left\{ \frac{\sum_{i=1}^k l(MST(S_i))}{l(G)} \mid l : E \to \mathbb{R}_{\geq 0},\ l \not\equiv 0 \right\}.$$

Corollary 1. $OPT \geq \frac{\sum_{i=1}^k l(MST(S_i))}{l(G)}$ *for all* $l : E \to \mathbb{R}_{\geq 0}$.

By complementary slackness, feasible solutions to (LP) and (LP*) are optimal if and only if

$$l(e)(c(e) - z) = 0 \text{ for all } e \in E \tag{1}$$

$$c_i(T)(l(T) - Y_i) = 0 \text{ for all } i \in \{1,\ldots,k\} \text{ and all } T \in \mathcal{T}_i \tag{2}$$

$$Y_i \left(\sum_{T\in\mathcal{T}_i} c_i(T) - 1 \right) = 0 \text{ for all } i \in \{1,\ldots,k\}. \tag{3}$$

To guarantee optimality, it is sufficient to replace (1), (2), and (3) by

$$l(e)(c(e) - c_{\max}) = 0 \text{ for all } e \in E, \tag{1'}$$

$$c_i(T)(l(T) - l(\text{MST}(S_i)) = 0 \text{ for all } i \in \{1,\ldots,k\}, \text{ and all } T \in \mathcal{T}_i, \tag{2'}$$

$$\sum_{T\in\mathcal{T}_i} c_i(T) = 1. \tag{3'}$$

Summing $(1')$ over all edges and $(2')$ over all $i \in \{1,\ldots,k\}$ and all $T \in \mathcal{T}_i$ yields

$$\sum_{e\in E} c(e)l(e) = l(G)c_{\max} \tag{1”}$$

$$\sum_{i=1}^{k} l(\mathrm{MST}(S_i)) = \sum_{i=1}^{k}\sum_{T\in\mathcal{T}_i} c_i(T)l(T) = \sum_{e\in E} c(e)l(e). \tag{2”}$$

In the derivation of $(2'')$ we used $(3')$ and the fact that

$$\sum_{i=1}^{k}\sum_{T\in\mathcal{T}_i} c_i(T)l(T) = \sum_{e\in E} l(e)\sum_{i=1}^{k}\sum_{\substack{T\in\mathcal{T}_i\\ e\in E(T)}} c_i(T)$$

which equals $\sum_{e\in E} l(e)c(e)$ by definition of $c(e)$. Any approximately optimal solution to (LP) that is found in polynomial time determines for each multicast request S_i a (polynomial) set of trees with fractional congestion. The task of selecting one of these trees for each S_i such that the maximum congestion is minimized constitutes a vector selection problem. Raghavan [18] bounds the quality of a solution by analyzing a randomized rounding procedure. For our problem his analysis gives the following result.

Theorem 1. *There exists an $O(k\beta\varepsilon^{-2}\ln k\ln m)$-time randomized algorithm for computing a solution to the minimum multicast congestion problem with congestion bounded by*

$$\begin{cases} (1+\varepsilon)rOPT + (1+\varepsilon)(\exp(1)-1)\sqrt{rOPT\cdot\ln m} & \text{if } rOPT \geq \ln m \\ (1+\varepsilon)rOPT + \frac{(1+\varepsilon)\exp(1)\ln m}{1+\ln(\frac{\ln m}{rOPT})} & \text{otherwise.} \end{cases}$$

2.5 Approximating the Fractional Optimum

Plotkin, Shmoys and Tardos specify conditions that are sufficient to guarantee relaxed optimality. Adapted to the multicast problem, these conditions are the following relaxations of $(1'')$ and $(2'')$.

$$(1-\varepsilon)c_{\max}l(G) \leq \sum_{e\in E} c(e)l(e) \tag{R1}$$

$$(1-\varepsilon)\sum_{e\in E} c(e)l(e) \leq \sum_{i=1}^{k} l(\tilde{\mathrm{MST}}(S_i)) + \varepsilon c_{\max}l(G). \tag{R2}$$

Their idea is to choose l such that (R1) is automatically satisfied, and to gradually satisfy (R2) by repeated calls to the following routine:
given a feasible solution $(c_i(T), c_{max})$ to (LP) and an error parameter ε, we define a length function l on the edges by $l(e) := \exp(\alpha\cdot c(e))$, where α depends on the maximum congestion c_{max}, the number of edges m and the parameter ε. Now, for each multicast request, an approximation $\tilde{T}_i$ of the Steiner minimum

tree is computed with respect to l. Our current solution is then modified by shifting congestion towards the computed Steiner trees, i.e., by setting

$$c_i(T) := \begin{cases} (1-\sigma)c_i(T) + \sigma & \text{for } T \in \{\tilde{T}_i \mid i \in \{1, \ldots, k\} \\ (1-\sigma)c_i(T) & \text{otherwise.} \end{cases}$$

Thus we punish the repeated choice of existing tree edges and gradually construct a fractional solution that approximates the optimum up to $(1+6\varepsilon)$ times the presently best factor for the Steiner tree approximation. It can be shown that (R1) is satisfied for $\alpha \geq \frac{2\ln(2m\varepsilon^{-1})}{c_{max}\varepsilon}$ and that the choice of $\sigma := \frac{5\varepsilon}{9\alpha k}$ guarantees that $l(G)$ decreases by a factor of least $(1 - \frac{5\varepsilon^2 c_{max}}{9k})$. Since the initial value of $l(G)$ is at most $m \cdot \exp(\alpha \cdot c_{max})$ we get the following result.

Theorem 2. *For $\varepsilon \leq \frac{1}{6}$ and assuming $OPT \geq 1$, the fractional minimum multicast congestion problem can be approximated within $r(1+6\varepsilon)OPT$ in time $O(k^2\varepsilon^{-2} \cdot \ln(m/\varepsilon)\beta)$. Using the Steiner tree approximation by Mehlhorn [19], we obtain a $2(1+6\varepsilon)$-approximation in time $\tilde{O}(k^2\varepsilon^{-2}m)$.*

We observed that the efficiency of the algorithm heavily depends on the way, the improvement of the congestion is implemented. By Theorem 2 it would be sufficient to fix an $\varepsilon > 0$ once and run the above described procedure. Practical experiments show that much better solutions are obtained, if we iterate the procedure starting with $\varepsilon := 1/6$, and decreasing ε in subsequent iterations by a factor of 1/2. This process is called ε-scaling.

Adapting the algorithm of Radzik [11] to our problem, yields a theoretical speedup by a factor of k. The key idea is to update the length function after each single approximate Steiner tree computation, if the length of the new tree is sufficiently small. Radzik uses relaxed optimality conditions different from (R1) and (R2). One can show that for $\lambda \geq$ OPT and $\varepsilon \leq 1/3$, we can guarantee $c_{max} \leq r(1+\varepsilon)$OPT if

$$c_{\max} \leq (1+\varepsilon/3)\lambda \text{ and} \tag{R1'}$$

$$\sum_{i=1}^{k} l(\tilde{\text{MST}}(S_i)) \geq (1-\varepsilon/2)\lambda l(G). \tag{R2'}$$

The length function of Radzik is very similar to the one of Plotkin et al.

Definition 1. *Given some fixed $\lambda \geq OPT$ we define*

$$\alpha := \frac{3(1+\varepsilon)\ln(m\varepsilon^{-1})}{\lambda\varepsilon} \textit{ and } l(e) := \exp(\alpha \cdot c(e)) \textit{ for all } e \in E.$$

we can prove the following result.

Theorem 3. *For $\varepsilon \leq \frac{1}{24}$, the fractional minimum multicast congestion problem can be approximated within $r(1+\varepsilon)OPT$ in time*

$$O\left(\varepsilon^{-2} \cdot k \ln n \ln k(m+\beta)\right) = \tilde{O}(km\varepsilon^{-2})$$

for the Steiner tree approximation by Mehlhorn.

The algorithm of Garg and Könemann [12] differs from the previous algorithms in the fact that there is no transfer of congestion old Steiner trees to new Steiner trees in successive iterations. Instead, the functions describing the congestion are built up from scratch. A feasible solution is obtained by scaling the computed quantities in the very end. Since the algorithm is quite simple, we give a detailed description.

```
ImproveCongestion
for $e \in E$
    $l(e) := \delta$
while $\sum_{e \in E} l(e) < \xi$
    for $i := 1$ to $k$
        $T := \tilde{\mathrm{M}}\mathrm{ST}(S_i)$, $c_i(T) := c_i(T) + 1$
        for $e \in T$
            $l(e) := l(e) \cdot \left(1 + \frac{\varepsilon}{u}\right)$
scale $\bar{c}_i(T) := c_i(T)/\#\mathrm{iterations}$
```

The following lemmas are needed for the analysis.

Lemma 2. *Suppose that $l(e)$ is initialized to some constant δ for all $e \in E$. Let q be the number of iterations of the* `while`*-loop* ImproveCongestion *needs to terminate. Then $\bar{c}^{(q)}_{\max} < \frac{u \cdot \ln(\xi/\delta)}{(q-1) \cdot \ln(1+\varepsilon)}$ and $q \leq \left\lceil \frac{u}{OPT} \cdot \frac{\ln(\xi/\delta)}{\ln(1+\varepsilon)} \right\rceil$*

To achieve an $r(1+\varepsilon)$OPT-approximation, we must fix the free parameters δ, ξ and u appropriately.

Lemma 3. *Let $\varepsilon \leq \frac{1}{36}$. For $u < 2OPT$ and $\xi := \delta \cdot \left(\frac{m}{1-2\varepsilon r}\right)^{\frac{1+\varepsilon}{\varepsilon}}$ the algorithm terminates with $\bar{c}_{\max} < r(1+6\varepsilon)OPT$.*

Altogether we have the following result.

Theorem 4. *An $r(1+6\varepsilon)OPT$-approximate solution to the fractional minimum multicast problem can be obtained in time $O(\beta\varepsilon^{-2}k \cdot \ln k \ln m) = \tilde{O}(\varepsilon^{-2}km)$ using the Steiner tree approximation by Mehlhorn.*

The proof is based on repeatedly calling ImproveCongestion in the following u-scaling process. Throughout the process we maintain the condition $u < 2$OPT to make sure that the algorithm terminates with the claimed guarantee. So all we have to worry about is the number of iterations. The first call is performed with $u := \frac{k}{2}$. By Lemma 3, if $u \leq$ OPT, the algorithm terminates within $\left\lceil \frac{\ln(\xi/\delta)}{\ln(1+\varepsilon)} \right\rceil$ iterations. Otherwise, we know that OPT $< \frac{k}{2}$ and restart the algorithm with $u := \frac{k}{4}$. Again, ImproveCongestion either terminates in $\left\lceil \frac{\ln(\xi/\delta)}{\ln(1+\varepsilon)} \right\rceil$ iterations or we may conclude that OPT $< \frac{k}{4}$ and restart the algorithm with $u := \frac{k}{8}$. Repeating this at most $O(\ln k)$ times yields the claim, since $\frac{\ln(\xi/\delta)}{\ln(1+\varepsilon)} = O(\varepsilon^{-2} \ln m)$

and because one iteration takes $O(k(m+\beta))$-time. Unfortunately, the above u-scaling process is *not practically efficient*, since the number of iterations in each scaling phase is considerably large (note that we need $\varepsilon \leq \frac{1}{36}$ to guarantee the approximation quality, so even for moderate values of m, say $m := 500$, we may have to await over 8000 iterations before we can decide whether or not OPT $< u$). However, without u-scaling the running time increases by a factor of $k/\ln k$. Again, we may (practically) improve the running time by using $\bar{c}_{\max} < (1+6\varepsilon)\sum_{i=1}^{k} l(\tilde{\mathrm{M}}\mathrm{ST}(S_i))/l(G)$ instead of $l(G) \geq \xi$ as the termination criterion. It is an open question whether or not ε-scaling can be advantageously applied. (Our experiments strongly indicate that ε-scaling substantially decreases the running time, but an analysis is still missing.) The following implementation takes care of the fact that due to limited precision it may be necessary to rescale the edge lengths.

```
ApproximateCongestion
$maxdual := 0$, $minprimal := k$, $c_{\max} := 0$, $iteration := 0$, $LG := m$
for $e \in E$

    $c(e) := 0$, $l(e) := 1$
    do $LM := 0$
        for $i := 1$ to $k$
            $T_i := \tilde{\mathrm{M}}\mathrm{ST}(S_i)$, $c(T_i) := c(T_i) + 1$
            for $e \in E(T_i)$
                $LG := LG - l(e)$
                $l(e) := l(e) \cdot \left(1 + \frac{\varepsilon}{k}\right)$, $c(e) := c(e) + 1$
                $c_{\max} := \max\{c_{\max}, c(e)\}$
                $LG := LG + l(e)$, $LM := LM + l(e)$
        $iteration := iteration + 1$, $dual := \frac{LM}{LG}$
        $minprimal := \min\{minprimal, \frac{c_{\max}}{iteration}\}$
        $maxdual := \max\{maxdual, dual\}$
        if $LG > 100000$
            $LG := \frac{LG}{100000}$
            for $e \in E$
                $l(e) := \frac{l(e)}{100000}$
    while $minprimal \geq (1 + 6\varepsilon) maxdual$

$c_{\max} := \frac{c_{\max}}{iteration}$,
for $T \in \hat{\mathcal{T}}_i$

    $c(T) := \frac{c(T)}{|\hat{\mathcal{T}}_i|}$
```

2.6 Approximating the Integral Optimum

Klein et al. [9] describe an approximation algorithm for the unit capacity concurrent flow problem that can be modified to approximate an integral solution to our problem without the necessity to round. The algorithm is similar to the approach of Plotkin, Shmoys and Tardos. However, instead of updating all trees in one iteration of the while loop, only one "bad" tree per iteration is modified.

Definition 2. *A tree $T \in \mathcal{T}_i$ is r-bad for S_i if*

$$(1-\varepsilon')l(T) > rl(MST(S_i)) + \varepsilon' \cdot \frac{c_{\max} \cdot l(G)}{k}.$$

This allows us to choose σ by a factor of k larger than in the algorithms described previously, and it can be shown that consequently σ never needs to be reduced below 1, if $\mathrm{OPT} = \Omega(\log n)$.

Theorem 5. *An integral solution to the minimum multicast congestion problem with congestion $OPT \cdot O(\sqrt{OPT \cdot \log n})$ can be found in time $O(k^2(m+\beta)\log k)$.*

A deterministic approximation satisfying the bounds of Theorem 1 can be proved with the tools of Srivastav and Stangier [16]. The proof given in [16] has to be slightly modified, as we have to replace the Angluin-Valiant inequality by Raghavan's stronger bound on the deviation of the sum of Bernoulli trials from its expected value [18]. The crucial difference is that in computing the pessimistic estimators we require an approximate solution to the equality

$$\frac{1}{m} = \left(\frac{\exp(\delta)}{(1+\delta)^{(1+\delta)}}\right)^{\mathrm{OPT}},$$

i.e. we want to determine δ such that $\left(\frac{\exp(\delta)}{(1+\delta)^{(1+\delta)}}\right)^{\mathrm{OPT}} \in \left[\frac{1}{2m}, \frac{1}{m}\right]$ or equivalently $0 \le f(\delta) := \ln(2m) + \mathrm{OPT}(\delta - (1+\delta)\ln(1+\delta)) \le \ln 2$. Note that f is a monotone function of δ with $f(0) = \ln 2 + \ln m > \ln 2$ and $f(6 + \ln 2m) \le \ln(2m) + (6 + \ln(2m) - (7 + \ln(2m)) \cdot 2) \le -8$. Hence, an approximate solution can be found in time $O(\ln \ln m)$ by binary search. In practical experiments the following algorithm was superior to all theoretically efficient approximation algorithms.

The algorithm uses different length functions for determining a start set of Steiner trees and for updating trees. The first length function is similar to the one in Aspnes [15] et al.'s online algorithm, but uses a base of n instead of a base $\in (1, 1.5]$. In fact, we found that the quality of the solutions increases substantially as the base grows. To improve near optimal solutions it turned out to be of advantage to replace the exponent $c(e)/c_{\max}$ in the length function by $c(e) - c_{\max}$, thereby increasing the impact of highly congested edges. On the other hand, the exponential base must be neither too small nor too large to "correctly" take into account edges with lower congestion. Here, the size of the current best tree (experimentally) proved to be a good choice.

```
PracticalCongestion
$\lambda := 1,\ c(e) := 0$ for all $e \in E$
for $i := 1$ to $k$

    for $e \in E$
        $l(e) := n^{\frac{c(e)}{\lambda} - 1}$
    $T_i := \widetilde{\text{MST}}(S_i)$
    for $e \in E(T_i)$
        $c(e) := c(e) + 1,\ \lambda = \max\{\lambda, c(e)\}$

$iteration := 1$
while ($iteration \leq 100$)

    for $i := 1$ to $k$
        $A := |E(T_i)|$
        for $e \in E$
            $l(e) := A^{c(e) - c_{\max}}$
        for $e \in E(T_i)$
            $l(e) := l(e)/A$
        $T_i' := \widetilde{\text{MST}}(S_i)$
        for $e \in E(T_i)$
            $l(e) := l(e) \cdot A$
        if $l(T_i) > l(T_i')$ then $T_i := T_i'$
        update $c(e)$
    $iteration := iteration + 1$
```

2.7 Open Problems

We tested our new algorithm against Aspnes et al.'s online algorithm and a version of ApproximateCongestion3, which is (together with Approximate-Congestion2) theoretically fastest[1] and found that our heuristic algorithm PrcaticalCongestion showed a superior performance. A proof confirming the experimental results remains a challenging open problem.

3 Construction of Sparse Asymmetric Connectors

3.1 Previous Work

Let $e(n, N, k)$ denote the minimum number of edges sufficient for building an (n, N)-connector of depth k, i.e., a connector between n inputs and N outputs where each input-output path has length k. The size of $e(n, N, k)$ in the

[1] Our test instances were grid graphs with rectangular holes. Such grid graphs typically arise in the design of VLSI logic chips where the holes represent big arrays on the chip. They are considered as hard instances for path- as well as for tree-packing problems. Multicast requests of size 50 to 1000 were chosen by a random generator.

symmetric case $N = n$ is well-studied. Pippenger and Yao [20] proved that $e(n, n, k) = \Omega(n^{1+1/k})$ and showed by a probabilistic argument that $e(n, n, k) = O(n^{1+1/k}(\log n)^{1/k})$. The best explicit construction of sparse symmetric connectors with odd depth is also due to Pippenger [21] who showed how to build $(n, n, 2j+1)$-connectors with $O(n^{1+1/(j+1)})$ edges. Hwang and Richards [22] gave a construction of an $(n, n, 2)$-connector with $O(n^{5/3})$ edges that can be extended by Pippenger's method to yield $(n, n, 2j)$-connectors with $O(n^{1+2/(3j-1)})$ edges $(j \geq 2)$. Less results are known for asymmetric connectors. Oruç [23] gave constructions of $(n, N, \Omega(\log_2 N+\log_2 n))$-connectors and $(n, N, \Omega(\log_2 N+\log_2^2 n))$-connectors with $O((N+n)\log_2 n)$ and $O(N+n\log_2 n)$ edges, respectively, relying on recursive Clos networks [24] and concentrators as building blocks. Richards and Hwang [25] were the first to construct asymmetric connectors of depth 2. Their method requires $N\lceil\sqrt{n+5/4}-1/2\rceil + n\lceil\sqrt{n+5/4}-1/2\rceil\sqrt{N}$ edges if $n \leq \sqrt{N}$, and $O(N^{3/4}n)$ edges if $n > \sqrt{N}$ provided that N is the square of a prime.

3.2 Our Results

We are interested in (n, N)-connectors with constant depth k and $n \ll N$. Note that for $k = 1$ the only solution is the complete bipartite graph $K_{n,N}$ (so-called *crossbar*). In practical applications (design of electronic switches) one typically has $n \leq \sqrt{N}$. After introducing a general structure for connectors, we determine the size of an optimal (n, N)-connector of depth 2 for $n \leq N^{1/2-\varepsilon}$ as $\Theta(N)$. As our main result we give explicit constructions for $(n, N, 2)$-connectors with at most $N\lceil\sqrt{3n/4}\rceil + n\lceil\sqrt{3Nn}\rceil$ edges. This improves over the complexity of the switching network of Richards and Hwang by a factor of $\sqrt{3/4}$. The constructions are based on a nice generalization of the Erdős-Heilbronn theorem on the size of restricted sums in the group $\mathbb{Z}_p$ by Lev [26]. Finally we mention results obtained jointly with Amnon Ta-Shma on the construction of near optimal connectors by means of expander techniques.

3.3 Sparse $(n, N, 2)$-Connectors

We will show that the following general structure is optimal for depth-2 connectors with $n \leq \sqrt{N}$:

- input vertices and intermediate vertices form a complete bipartite digraph,
- each output is connected to d intermediate vertices for some constant d.

This structure requires $n \cdot |L| + N \cdot d$ edges. Let us call a digraph with the above structure a *standard switch*. For a standard switch, the problem of finding vertex disjoint paths is reduced to that of determining a perfect matching between intermediate and output vertices. Moreover, Hall's [27] marriage theorem (assuring that a bipartite graph $G = (A, B, E)$ contains a matching of A if and only if each subset S of A has at least $|S|$ neighbors in B) provides a handy connector condition. For $S \subseteq V$ let $\Gamma(S)$ denote the set of neighbors of S.

Proposition 1 (connector condition). *A standard switch is an $(n, N, 2)$ connector if $|\Gamma(S)| \geq |S|$ for all $S \subseteq O$ with $|S| \leq n$.*

Proof. Let $f : I \to O$ be injective. Since the connector condition is satisfied we can apply Hall's theorem to prove that $f(I)$ can be perfectly matched into L via n disjoint edges $e_1, \ldots, e_n$, where $e_i = (v_i, f(i))$ for some intermediate vertex $v_i \in L$, $i = 1, \ldots, n$. I and L form a complete bipartite graph, thus $\{(i, v_i, f(i)) \mid i \in I\}$ is the desired set of vertex disjoint paths. □

Richards and Hwang used the standard switch structure for their construction in the case $n \leq \sqrt{N}$. The following theorem is implicit in their paper.

Theorem 6 (Richards, Hwang, 1985). *Let $n, N \in \mathbb{N}$ with $n \leq \sqrt{N}$ and let p be the smallest prime $\geq \sqrt{N}$. The construction of Richards and Hwang yields a connector with at most $N \cdot \left\lceil \sqrt{n + \frac{5}{4}} - \frac{1}{2} \right\rceil + p \left\lceil \sqrt{n + \frac{5}{4}} - \frac{1}{2} \right\rceil n$ edges.*

3.4 An Optimal $(n, N, 2)$-Connector

Theorem 7. *Let $n \leq N^{\frac{1}{2}-\varepsilon}$ for some $0 < \varepsilon < 1/2$. An optimal (n, N)-connector of depth 2 has $\Theta(N)$ edges.*

Proof. The proof of the lower bound is trivial, since each of the N output vertices must have degree ≥ 1. For the proof of the upper bound we assume $\sqrt{N}$ to be integral w.l.o.g. Consider a standard switch where $|L| = \sqrt{N}$. For each output vertex pick d intermediate vertices as neighbors uniformly at random. Let us determine d such that the construction is successful. If Hall's condition is not satisfied, there is a subset $S \in O$ of size at most n whose neighborhood is smaller than $|S|$. Consequently, there must be no edges connecting S to the $|L| - |\Gamma(S)|$ remaining intermediate vertices. The probability of these bad events is bounded by

$$\sum_{i=1}^{n} \binom{N}{i} \binom{\sqrt{N}}{\sqrt{N} - i} \cdot \left(1 - \frac{\sqrt{N} - i}{\sqrt{N}}\right)^{di} \leq \sum_{i=1}^{n} N^{2i} \cdot \left(\frac{n}{\sqrt{N}}\right)^{di}.$$

We have to ensure that the latter term is smaller than 1, which is true for $N^2 \cdot \left(\frac{n}{\sqrt{N}}\right)^d = \frac{1}{2}$ and thus for

$$d = \frac{\log(2N^2)}{\log \sqrt{N} - \log n} \leq \frac{2 \log N + \log 2}{\frac{1}{2} \cdot \log N - (\frac{1}{2} - \varepsilon) \log N} = \frac{2}{\varepsilon} + \frac{\log 2}{\varepsilon \log N} = O(1).$$

So, there is a connector with $N \cdot d + \sqrt{N} \cdot N^{\frac{1}{2}-\varepsilon} = O(N)$ edges. □

3.5 Explicit Constructions

The following basic method assigns each output two neighbors in some small subset of L and subsequently copies this neighborhood several times until the connector condition is satisfied. In case of odd d a further neighbor is added to the last copy.

2-Copy Method

1. Let $O := \{0, \dots, N-1\},\ q := \left\lceil \sqrt{N} \right\rceil,\ d := \left\lfloor \frac{2n-2}{\lceil 2\sqrt{n} \rceil} + 1 \right\rfloor$.
2. Let $L := \{0, \dots, qd-1\}$ (still, we think of L and O as disjoint sets).
3. For $x \in O$ let $x = x_1 + qx_2$ $(x_1, x_2 \in \{0, \dots, q-1\})$ be the q-ary decomposition of x.
4. Let $\Gamma_0(x) := \{x_1, x_2 + q\}$ and choose the neighbors of x as

$$\Gamma(x) := \bigcup_{i=0}^{\lfloor \frac{d}{2} - 1 \rfloor} (\Gamma_0(x) + i \cdot 2q).$$

If d is odd, add $(x_1 + x_2) \pmod q + (d-1)q$ as a further neighbor.

Theorem 8. *The 2-copy method yields a connector with at most $N\lceil\sqrt{n}\rceil + \lceil\sqrt{N}\rceil\lceil\sqrt{n}\rceil n$ edges.*

The proof is based on verifying Hall's condition for the neighborhoods constructed by the copy method. If N is *not* the square of a prime, and $n \cdot |L|$ is the dominating term for the number of edges, then the 2-copy method can readily improve over the construction of Richards and Hwang. A *general* improvement can be obtained by exploiting the following observation: the lower bound of $2\sqrt{|S|}$ for $|\Gamma_0(S)|$ is attained if and only if $|f_1(S)| = |f_2(S)| = \sqrt{|S|}$. However, in this case, $|f_1(S) + f_2(S)| \geq 2\sqrt{|S|} - 1$ by the famous Theorem of Cauchy and Davenport.

Theorem 9 (Cauchy 1813, Davenport 1935). *Let p be a prime; let A be a set of k different residue classes $\pmod p$ and let B be a set of l different residue classes $\pmod p$. Then the number z of residue classes $\pmod p$ expressible as $a + b$ $(a \in A,\ b \in B)$ satisfies*

$$z \geq \min\{p, k + l - 1\}.$$

Hence, $|f_1(S)| + |f_2(S)| + |f_1(S) + f_2(S)|$ should always be strictly larger than $\frac{3}{2}\left\lceil 2\sqrt{|S|} \right\rceil$, which was our bound for $|\Gamma_1(S)|$ in case d is odd. Therefore it is promising to modify the 2-copy method into a 3-copy method by replacing step 4 with

4' Let $\Gamma_0(x) := \{x_1, x_2 + q, (x_1 + x_2) \pmod q + 2q\}$ and choose the neighbors of x as

$$\Gamma(x) := \bigcup_{i=0}^{\lfloor \frac{d}{3} - 1 \rfloor} (\Gamma_0(x) + i \cdot 3q).$$

If $d \equiv 1 \bmod 3$, add $x_1 + (d-1)q$ as a further neighbor. If $d \equiv 2 \bmod 3$, add the neighbors $x_1 + (d-2)q$ and $x_2 + (d-1)q$.

It can be shown that for the 3-copy method it is enough to take d as $\left\lceil\sqrt{\frac{3}{4}n}\right\rceil + c$, $c \geq 2$ at the expense of a slightly larger $q := \min\{q' \in \mathbb{P} \mid q' \geq \sqrt{N}\}$. The proof is based on the following theorem of Lev [26] which is a generalization of the long-standing open Erdős-Heilbronn conjecture (resolved affirmatively by da Silva and Hamidoune in 1994).

Theorem 10 (Lev, 2000). *Let p be a prime, let $A, B \subseteq \mathbb{Z}/p\mathbb{Z}$ with $a := |A|, b := |B|$, and let $\mathcal{R} \subseteq A \times B$. Assume for definiteness $a \leq b$, and let $r := |\mathcal{R}|$. Then*

$$|A \overset{\mathcal{R}}{+} B| \geq \begin{cases} a + b - 2\sqrt{r} - 1, & \text{if } a + b \leq p + \sqrt{r} \text{ and } \sqrt{r} \leq a, \\ p - \frac{r}{a+b-p}, & \text{if } a + b \geq p + \sqrt{r}, \\ b - \frac{r}{a}, & \text{if } \sqrt{r} \geq a, \end{cases}$$

where $A \overset{\mathcal{R}}{+} B := \{a + b \mid a \in A, b \in B, (a, b) \notin \mathcal{R}\}$.

Verifying the claimed improvement requires some tedious case distinctions. The following *summation method* is simpler than the 3-copy method and also requires only $N\left\lceil\frac{3}{4}n\right\rceil + 2n\lceil\sqrt{3n/4}\rceil\lceil\sqrt{N}\rceil$ edges.

Let $N \geq 3n$.

1. Let $O = \{0, \ldots, N-1\}$, $q := \left\lceil\sqrt{N}\right\rceil$, $p := \min\{p' \in \mathbb{P} \mid p' \geq q\}$.
2. Let $d := \begin{cases} \left\lfloor \frac{2n-2}{\lceil 2\sqrt{n}\rceil} + 1 \right\rfloor, & n \leq 7 \\ \left\lfloor \frac{3n-3}{\lceil 2\sqrt{3n}\rceil} + 1 \right\rfloor, & n \geq 8 \end{cases}$, $L := \{0, \ldots, pd-1\}$.
3. For each $x \in O$ let $x = x_1 + qx_2$ be the q-ary decomposition of x. Define mappings $f_i : O \to \{0, \ldots, p-1\}$, $x \mapsto (x_1 + ix_2) \pmod p$ for all $i \in \{0, \ldots, d-1\}$.
4. Choose the neighborhood of $x \in O$ as $\Gamma(x) := \bigcup_{i=0}^{d-1} f_i(x) + i \cdot p$ $(\subseteq L)$.

Theorem 11. *The summation method yields a connector with at most*

$$N\left\lceil\sqrt{\frac{3}{4}n}\right\rceil + 2n\left\lceil\sqrt{3n/4}\right\rceil\left\lceil\sqrt{N}\right\rceil n$$

edges.

Proof. If $n \leq 7$, the summation method is similar to the 2-copy method. So we may actually apply the 2-copy method in this case and thus have to consider only the case $n \geq 8$ in our proof. For $n \geq 8$ we have $d \geq 3$, so the connector condition holds for all $S \subseteq \{0, \ldots, N-1\}$ with $|S| \leq 3$. Choose $S \subseteq \{0, \ldots, N-1\}$ with $|S| =: s \geq 4$. Let $\mathcal{D}_3$ abbreviate the set of all three element subsets of $\{0, \ldots, d-1\}$. We are going to show that for each $J \in \mathcal{D}_3$,

$$\sum_{j \in J} |f_j(S)| \geq \left\lceil 2\sqrt{3s}\right\rceil. \tag{4}$$

From (4) we conclude that

$$\begin{aligned}|\mathit{\Gamma}(S)| &= \sum_{j=0}^{d-1} |f_j(S)| = \frac{1}{\binom{d-1}{2}} \sum_{J\in\mathcal{D}_3} \sum_{j\in J} |f_j(S)| \\ &\geq \frac{\binom{d}{3}}{\binom{d-1}{2}} \left\lceil 2\sqrt{3s}\right\rceil = \frac{d}{3}\left\lceil 2\sqrt{3s}\right\rceil \\ &> \frac{n-1}{\left\lceil 2\sqrt{3n}\,\right\rceil} \cdot \left\lceil 2\sqrt{3s}\right\rceil \geq \frac{s-1}{\left\lceil 2\sqrt{3s}\,\right\rceil} \cdot \left\lceil 2\sqrt{3s}\right\rceil \\ &= s-1,\end{aligned}$$

and hence $|\mathit{\Gamma}(S)| \geq s$. Thus, the summation method yields a connector. Since

$$\begin{aligned}\left\lfloor \frac{3n-3}{\left\lceil 2\sqrt{3n}\right\rceil} + 1 \right\rfloor \leq \frac{3n-3}{2\sqrt{3n}} + 1 &= \sqrt{\frac{3}{4}n} - \sqrt{\frac{3}{4n}} + 1 \\ &< \sqrt{\frac{3}{4}n} + 1\end{aligned}$$

we have $d \leq \left\lceil \sqrt{\frac{3}{4}n}\right\rceil$ which gives us the claimed bound on the number of edges (the second term in the estimate uses Bertrand's theorem which states that there is always a prime between x and $2x$ for all $x \in \mathbb{N}$).

It remains to prove (4). For this, let $J = \{j_1, j_2, j_3\} \subseteq \{0, \ldots, d-1\}$, and set $\lambda_1 := j_3 - j_2$, $\lambda_2 := j_3 - j_1$, $\lambda_3 := j_2 - j_1$. We have

$$|\lambda_i f_{j_i}(S)| = |f_{j_i}(S)| \text{ for } i \in \{1,2,3\} \tag{5}$$

and

$$\begin{aligned}\lambda_2 f_{j_2}(S) &= \{(j_3 - j_1)(x_1 + j_2 x_2) \pmod p \mid x \in S\} \\ &= \{x_1(\lambda_1 + \lambda_3) + x_2(j_1\lambda_1 + j_3\lambda_3) \pmod p \mid x \in S\} \\ &= \{\lambda_1(x_1 + j_1 x_2) + \lambda_3(x_1 + j_3 x_2) \pmod p \mid x \in S\} \\ &= \lambda_1 f_{j_1}(S) \overset{\mathcal{R}}{+} \lambda_3 f_{j_3}(S),\end{aligned} \tag{6}$$

where $\mathcal{R} := f_{j_1}(S) \times f_{j_3}(S) \setminus \{(f_{j_1}(x), f_{j_3}(x)) \mid x \in S\}$. Similarly, $f_{j_1}(S)$ and $f_{j_3}(S)$ can be expressed as (multiples of) some $\mathcal{R}'$-restricted sum of (multiples of) $f_{j_2}(S)$, $f_{j_3}(S)$ and $f_{j_1}(S), f_{j_2}(S)$, respectively. Due to (5) and (6) we can use Theorem 10 to estimate $\sum_{j\in J} |f_j(S)|$. Let us abbreviate $a := |f_{j_1}(S)|$, $b := |f_{j_3}(S)|$, and $r := |\mathcal{R}| = ab - s$. Assuming that $a \leq b$ we have to distinguish three cases:

Case 1: $a + b \leq p + \sqrt{r}$, and $\sqrt{r} \leq a$
Let $c := |f_{j_2}(S)|$. By Theorem 10,

$$\sum_{j\in J} |f_j(S)| = a + b + c \geq 2(a+b) - 2\sqrt{r} - 1 = 2(a+b) - 2\sqrt{ab-s} - 1.$$

Since we seek for a worst case lower bound, we may assume that $a+b+c = 2(a+b) - 2\sqrt{ab-s} - 1$. The function $x \mapsto a+b+c = 2(a+b-\sqrt{ab-s}) - 1$ attains a unique minimum for $a = b = \sqrt{\frac{4}{3}s}$ (on $\mathbb{R}_{>0}$). Since $f_{j_3}(S)$ may be regarded as (the multiple of) the restricted sum of $f_{j_1}(S)$ and $f_{j_2}(S)$, we also have a unique minimum of $x \mapsto a+b+c = 2(a+c-\sqrt{ac-s}) - \alpha$ for some $\alpha \leq 1$ at $a = c = \sqrt{\frac{4}{3}s}$, and thus $\sum_{j\in J} |f_j(S)| \geq \left\lceil 3\sqrt{\frac{4}{3}s} \right\rceil = \lceil 2\sqrt{3s} \rceil$.

The remaining two cases $a+b \geq p+\sqrt{r}$ and $\sqrt{r} \geq a$ can be dealt with in a similar fashion. □

3.6 The Case $n > \sqrt{N}$

For the case $n > \sqrt{N}$ we give a construction with at most $(2+o(1))\, N^{\frac{3}{4}} n$ edges. For comparison, Richards and Hwang [25] gave a similar construction that can be shown to use at most $(3+o(1))N^{\frac{3}{4}}n$ edges.

Our construction is as follows. Again we assume $\sqrt{N}$ to be integral.

1. Construct a $(N, \sqrt{N}, 2)$-connector.
2. Let $m := \left\lceil \frac{n}{\sqrt{N}} \right\rceil$.
3. Copy all input vertices $I_1 := I$ and intermediate vertices $L_1 := L$ m times, so that we have input vertices $I_1, I_2, \ldots, I_m$ and intermediate vertices $L_1, L_2, \cdots, L_m$.
4. Connect the output vertices O with $L_2, \cdots, L_m$ in the same way as O is connected with L_1.
5. Connect L_j with I_j for $j = 2, \cdots, m$ in the same way as L_1 is connected with I_1.

Theorem 12. *For $n > \sqrt{N}$, the 2-copy method constructs connectors requiring at most $(2+o(1))\, N^{\frac{3}{4}} n$ edges.*

3.7 Sparse Asymmetric Connectors via Expander Techniques

In a joint work with Gerold Jäger and Amnon Ta-Shma, we gave an explicit construction of a sparse asymmetric connector with only $N\cdot$ poly$(\log N) + O(n^2)$ edges by means of expander graphs.

Definition 3. *Let $c \geq 1$ be a constant. A bipartite graph $G = (V_1, V_2; E)$ is an (n, c)-expander, if for every $S \subseteq O$ of cardinality at most n we have $|\Gamma(S)| \geq c \cdot |S|$.*

We were able to prove:

Theorem 13. *Let $c \geq 2$ be a constant. For every $n \leq N$ there exists an explicit construnction of a bipartite graph $G = (O, L : E)$ with $|O| = N$ and $|L| = 2^c$ that is an $(n, 2^{c-2})$-expander of degree poly($\log N$).*

This result was obtained by adapting the optimal disperser construction in [28] to the expander case. This involved replacing the extractor ingredient with so-called *universal extractors* from a previous construction of [29]. As an almost immediate corollary we got the following result.

Theorem 14. *For every $n \leq N$ there exists an explicit $(n, N, 2)$-connector with $N\cdot$ poly $(\log N) = O(n^2)$ edges. For $n \leq \sqrt{N}$ this ammounts to a connector with $N\cdot$ poly $(\log N)$ edges.*

Ta-Shma discovered, that one can build the unbalanced expanders we used more easily, by taking $O(\log N)$ copies of optimal *explicit dispersers* introduced in [28].

Definition 4 (Ta-Shma, Umans, Zuckerman, 2001). *A bipartite graph $G = (V_1, V_2; E)$ with $|V_1| = 2^n$, $|V_2| = 2^m$, and left degree 2^d is a $(2^k, \varepsilon)$-disperser if every subset $A \subseteq V$ of cardinality at least 2^k has at least $(1 - \varepsilon) \cdot 2^m$ distinct neighbors in W. A disperser is explicit if the i-th neighbor of vertex $v \in V_1$ can be computed in poly(n, d) time.*

Ta-Shma, Umans, and Zuckerman proved in [28] that for every n, k, and constant ε there is a degree poly(n) explicit $(2^k, \varepsilon)$-disperser $G = (V_1, V_2; E)$ with $|V_1| = 2^n$ and $|V_2| = \Omega(2^k \text{ poly}(n)/n^3)$. With these dispersers, the construction of the desired connectors is similar to a non-blocking network construction of Widgerson and Zuckerman [30]. The idea is to construct the connector as a graph $G = (V, E)$, $V = I \cup L \cup O$, where $G_{|_{I \cup L}}$ is a complete bipartite graph and $G_{|_{L \cup O}}$ is the disjoint union of $O(\log(N))$ graphs G_i, such that the i-th graph G_i is a $(2^{i+1}\cdot \text{ poly}(\log N), 1/2)$-disperser connecting the the output nodes to 2^{i+1} intermediate nodes.

3.8 Open Problems

1. The 2-copy method is based on the fact that any injective mapping from O to $\mathbb{N}^2$ induces a decomposition of $S \subseteq O$ into two sets A, B such that one is large if the other is small, assuring $|A| + |B| \geq \left\lceil 2\sqrt{|S|} \right\rceil$. By taking d mappings where each pair injectively maps O into $\mathbb{N}^2$ the bound generalizes to $\sum_{i=1}^{d} |A_i| \geq \left\lceil \frac{d}{2} \cdot \left\lceil 2\sqrt{|S|} \right\rceil \right\rceil$. One can easily give examples for which this bound is tight. The 3-copy method takes advantage of introducing the restricted sum $C := A \overset{\mathcal{R}}{+} B$, as a third set, which is large if both A and B are small, yielding $|A| + |B| + |C| \geq \left\lceil \sqrt{3|S|} \right\rceil$. A generalization to d sets is provided by the summation method: here, each set may be regarded as (a multiple of) the restricted sum of (multiples of) any two other sets, and thus $\sum_{i=1}^{d} |A_i| \geq \left\lceil \frac{d}{3} \left\lceil 2\sqrt{3|S|} \right\rceil \right\rceil$. Clearly, the next step would be to consider a "base" of four sets A, B, C, D, where D has a large lower bound if A, B, and C are simultaneously small, i.e. $|A| = |B| = |C| = \sqrt{\frac{4}{3}|S|}$. However, it is not clear how to construct these sets. Another promising way of improving the 3-copy method is the following: before taking the restricted sum, we map A and B onto some specially structured sets A', B' where $A' \overset{\mathcal{R}}{+} B'$ is larger than guaranteed by Theorem 4. An extremal example would be mapping A and B onto Sidon sets A', B', so that $|A' \overset{\mathcal{R}}{+} B'| = |S|$. Of course, using Sidon sets increases the number of intermediate vertices to N, making the

construction useless. Still, a proper weakening of Sidon sets might give a significant improvement.

2. Furthermore, it would be interesting to have efficient constructions for the multicast case where each of the n inputs is allowed to connect to $r \geq 2$ specified outputs.
3. Finally, a challenging open problem is concerned with derandomizing the probabilistic proof of Theorem 7.

Acknowledgments

Ahlswede and Aydinian [31] gave a nice construction based on the theorem of Kruskal and Katona using only $(1 + o(1)) \cdot N \log_2(N)$ edges in case $n < N^{1/\sqrt{\log_2 N}}$.

We would like to thank Amnon Ta-Shma for the idea of the probabilistic proof of Theorem 7.

References

1. Baltz, A., Srivastav, A.: Fast approximation of minimum multicast congestion - implementation versus theory. In: Petreschi, R., Persiano, G., Silvestri, R. (eds.) CIAC 2003. LNCS, vol. 2653, pp. 165–177. Springer, Heidelberg (2003)
2. Baltz, A., Srivastav, A.: Fast approximation of minimum multicast congestion - implementation versus theory. Operations Research 38(4), 319–344 (2004)
3. Baltz, A., Jäger, G., Srivastav, A.: Constructions of sparse asymmetric connectors with number theoretic methods. Networks 45(3), 119–124 (2005)
4. Vempala, S., Vöcking, B.: Approximating multicast congestion. In: Aggarwal, A.K., Pandu Rangan, C. (eds.) ISAAC 1999. LNCS, vol. 1741, pp. 367–372. Springer, Heidelberg (1999)
5. Carr, R.D., Vempala, S.: Randomized metarounding. Random Struct. Algorithms 20(3), 343–352 (2002)
6. Shahrokhi, F., Matula, D.W.: The maximum concurrent flow problem. J. ACM 37(2), 318–334 (1990)
7. Goldberg, A.V.: A natural randomization strategy for multicommodity flow and related algorithms. Inf. Process. Lett. 42(5), 249–256 (1992)
8. Leighton, F.T., Makedon, F., Plotkin, S.A., Stein, C., Stein, É., Tragoudas, S.: Fast approximation algorithms for multicommodity flow problems. J. Comput. Syst. Sci. 50(2), 228–243 (1995)
9. Klein, P.N., Plotkin, S.A., Stein, C., Tardos, É.: Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. SIAM J. Comput. 23(3), 466–487 (1994)
10. Plotkin, S.A., Shmoys, D.B., Tardos, É.: Fast approximation algorithms for fractional packing and covering problems. In: FOCS, pp. 495–504. IEEE, Los Alamitos (1991)
11. Radzik, T.: Fast deterministic approximation for the multicommodity flow problem. Math. Program. 77, 43–58 (1997)
12. Garg, N., Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: FOCS, pp. 300–309 (1998)

13. Grigoriadis, M.D., Khachiyan, L.G.: Approximate minimum-cost multicommodity flows in õ(epsilon^{-2}knm) time. Math. Program. 75, 477–482 (1996)
14. Jansen, K., Zhang, H.: Approximation algorithms for general packing problems with modified logarithmic potential function. In: Baeza-Yates, R.A., Montanari, U., Santoro, N. (eds.) IFIP TCS. IFIP Conference Proceedings, vol. 223, pp. 255–266. Kluwer, Dordrecht (2002)
15. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S.A., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. J. ACM 44(3), 486–504 (1997)
16. Srivastav, A., Stangier, P.: On complexity, representation and approximation of integral multicommodity flows. Discrete Applied Mathematics 99(1-3), 183–208 (2000)
17. Robins, G., Zelikovsky, A.: Improved steiner tree approximation in graphs. In: SODA, pp. 770–779 (2000)
18. Raghavan, P.: Probabilistic construction of deterministic algorithms: Approximating packing integer programs. J. Comput. Syst. Sci. 37(2), 130–143 (1988)
19. Magun, J.: Greedy matching algorithms: An experimental study. ACM Journal of Experimental Algorithms 3, 6 (1998)
20. Pippenger, N., Yao, A.C.C.C.: Rearrangeable networks with limited depth. SIAM Journal on Algebraic and Discrete Methods 3(4), 411–417 (1982)
21. Pippenger, N.: On rearrangeable and non-blocking switching networks. J. Comput. Syst. Sci. 17(2), 145–162 (1978)
22. Hwang, F., Richards, G.: A two-stage network with dual partial concentrators. j-networks (23), 53–58 (1992)
23. Oruç, A.Y.: A study of permutation networks: New designs and some generalizations. J. Parallel Distrib. Comput. 22(2), 359–366 (1994)
24. Clos, C.: A study of non-blocking switching networks. Bell System Tech. J. 32, 406–424 (1953)
25. Richards, G., Hwang, F.: A two-stage rearrangeable broadcast switching network. j-IEEE-com (33), 1025–1035 (1985)
26. Lev, V.F.: Restricted set addition in groups, ii. a generalization of the erdo"s-heilbronn conjecture. Electr. J. Comb. 7 (2000)
27. Diestel, R.: Graphentheorie, 2nd edn. Springer, New York (2000)
28. Ta-Shma, A., Umans, C., Zuckerman, D.: Loss-less condensers, unbalanced expanders, and extractors. In: STOC, pp. 143–152 (2001)
29. Raz, R., Reingold, O., Vadhan, S.P.: Extracting all the randomness and reducing the error in trevisan's extractors. J. Comput. Syst. Sci. 65(1), 97–128 (2002)
30. Wigderson, A., Zuckerman, D.: Expanders that beat the eigenvalue bound: Explicit construction and applications. Combinatorica 19(1), 125–138 (1999)
31. Ahlswede, R., Aydinian, H.K.: Construction of asymmetric connectors of depth two. J. Comb. Theory, Ser. A 113(8), 1614–1620 (2006)

Management of Variable Data Streams in Networks

Anja Feldmann[1], Simon Fischer[2], Nils Kammenhuber[3], and Berthold Vöcking[2]

[1] TU Berlin
[2] RWTH Aachen
[3] TU München
Anja.Feldmann@telekom.de,
{fischer,voecking}@cs.rwth-aachen.de,
hirvi@net.in.tum.de

Abstract. This work is motivated by the observation that traffic in large networks like the Internet is not controlled by a central authority but rather by a large number of selfish agents interacting in a distributed manner. Game theory predicts that selfish behaviour in such a system leads to a Nash equilibrium, but it does not, in general, explain, how such an equilibrium can be reached. We focus on this dynamic aspect.

In the first part of this survey, we develop a dynamic model of selfish, adaptive behaviour in routing networks. We show how and under which conditions such behaviour can give rise to a stable state and analyse the convergence time. Based on these results we design a distributed algorithm to compute approximate equlibria.

In the second part, we build upon the theory developed so far in order to design an online traffic engineering protocol which proceeds by adapting route weights on the time scale of seconds. We show that our protocol converges quickly and significantly improves network performance.

1 Introduction

The central aspect of the project “Management of Variable Data Strams in Networks” is the fact that traffic in large networks like the Internet does not follow a global plan prescribed by a central authority, but is the result of the interaction between numerous participants with different and possibly conflicting objectives. Our work is mainly concerned with static and dynamical analysis of network traffic based on game theoretical models, on the one hand, and on empirical studies of real Internet traffic, on the other hand.

Due to the selfishness and strategic behaviour of agents in Internet-like networks, game theory plays an increasingly important role in the analysis. The dominant solution concept is the notion of Nash equilibria which are states of a system in which no agent can gain by deviating unilaterally. Classical game theory requires various assumptions to motivate this concept, including complete knowledge and full rationality of the agents. Arguably, none of these assumptions are satisfied for the Internet. Here, we present an evolutionary approach which

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 266–291, 2009.

describes equilibria in Wardrop routing games as the outcome of an adaptive dynamic process by which agents revise their routing decisions at intervals.

One obstacle in describing such processes is the fact that agents may make concurrent rerouting decisions resulting in oscillating behaviour. This is a phenomenon well known in theory and practice. For a large class of rerouting dynamics we show how this behaviour can be avoided by introducing a suitable damping factor. In addition, we present a particular rerouting policy for which we can prove fast convergence. This policy behaves like a fully polynomial approximation scheme (FPAS) with respect to approximate equilibria.

The convergence results obtained so far can be applied in two ways. First, we derive from our policy a distributed algorithm for computing approximate equilibria. Second, we can build upon these results to design dynamic traffic engineering protocols. We present such a protocol and study its convergence properties in a realistic simulation model which is more detailed than the theoretic model, e. g., with respect to interactions of changes in user demands and TCP congestion control together with load-adaptive routing.

1.1 The Wardrop Model

In 1952, Wardrop [47] (for a good introduction see also [42]) proposed a model of selfish traffic in networks. An *instance* of the Wardrop routing game Γ is given by a finite directed multi-graph $G = (V, E)$, a family of differentiable, non-decreasing, non-negative, Lipschitz-continuous *latency functions* $(\ell_e)_{e \in E}$, $\ell_e : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$, and a family of k *commodities*, each specified by a source node $s_i \in V$, a target node $t_i \in V$, and a flow demand $r_i \in \mathbb{R}_+$ to be routed from s_i to t_i. The total flow demand is denoted by $r = \sum_{i \in [k]} r_i$. For $i \in [k]$, let $\mathcal{P}_i$ denote the set of acyclic paths in G that start at s_i and end at t_i, and let $\mathcal{P} = \bigcup_{i \in [k]} \mathcal{P}_i$. An instance is *single-commodity* if $k = 1$ and *multi-commodity* if $k > 1$.

A flow vector $(f_P)_{P \in \mathcal{P}}$ is *feasible* for an instance Γ if it is non-negative and satisfies the flow demands, i. e., $\sum_{P \in \mathcal{P}_i} f_P = r_i$ for all $i \in [k]$. Let $\mathcal{F}_\Gamma$ denote the polyhedron specified by these constraints. The path flow vector f uniquely determines edge flows $f_e = \sum_{P \ni e} f_P$ for all edges $e \in E$. A flow vector f induces latencies on the edges. The latency of edge $e \in E$ is $\ell_e(f) = \ell_e(f_e)$. Latency is additive along a path, i. e., for any path $P \in \mathcal{P}$, $\ell_P(f) = \sum_{e \in P} \ell_e(f)$. For each commodity $i \in [k]$ this implies an *average latency* of $L_i(f) = \sum_{P \in \mathcal{P}_i} \frac{f_P}{r_i} \cdot \ell_P(f)$. The average latency of the entire flow is then $L(f) = \sum_{P \in \mathcal{P}} \frac{f_P}{r} \cdot \ell_P(f)$. In the following, we normalise $r = 1$. Whenever we omit the argument f to some of the above quantities, we always refer to the current flow which will be clear from context.

A classical objective in this scenario is to find a feasible flow f that minimises the overall latency $L(f)$. Taking the game theoretic approach we are not seeking for an optimal, but rather for a stable flow. To that end, we consider the flow to be composed of infinitesimal contributions controlled by an infinite number of agents. Then, a flow is stable if none of these agents can improve their sustained latency by exchanging their path with another one. Such a flow vector can be characterised as follows [27]:

Definition 1 (Wardrop equilibrium). *A flow vector f is at a* Wardrop equilibrium *if for all commodities $i \in [k]$ and all $P_1, P_2 \in \mathcal{P}_i$ with $f_{P_1} > 0$ it holds that $\ell_{P_1}(f) \leq \ell_{P_2}(f)$. The set of Wardrop equilibria is denoted by $\mathcal{F}_\Gamma^{NE}$.*

It is well known that Wardrop equilibria minimise a potential function due to Beckmann, McGuire, and Winsten [6]. Let

$$\Phi(f) = \sum_{e \in E} \int_0^{f_e} \ell_e(u)\, du \ . \tag{1}$$

A flow is at a Wardrop equilibrium with respect to $(\ell_e)_{e \in E}$ if and only if it minimises Φ over all feasible flows. We denote the minimum potential by $\Phi^* = \min_{f \in \mathcal{F}_\Gamma} \Phi(f)$. The crucial property of this potential is that if a flow of dx is moved from path P to path Q, the potential changes by $(\ell_Q - \ell_P)\, dx$.

1.2 Related Work

Game Theory. A natural question concerning the Wardrop model asks for the degradation of performance due to the selfishness of the agents. To that end, one considers the *price of anarchy* $\rho(\Gamma)$ defined as the ratio between the total latency of a worst Wardrop equilibrium and the minimal total latency. For affine latency functions, Roughgarden and Tardos [42,40] show that $\rho = 4/3$. For polynomial latency functions with degree d, $\rho = \Theta(d/\log d)$.

In [41] the objective of minimising the maximum latency rather than the average is considered. Cominetti *et al.* [14] analyse games where users control a non-negligible amount of traffic which may be split among several paths. For a finite number of users with unsplittable flow, bounds on the price of anarchy are presented in [2,12]. In [28] the authors consider games in which groups of users may collaborate.

Several approaches have been proposed to decrease the price of anarchy. In [13] and [23] a scenario is considered in which taxes are imposed on the users such that Nash equilibria coincide with Wardrop equilibria, and bounds on the magnitude of such taxes are presented. Another possibility to decrease the price of anarchy is to control a fraction of the individuals centrally. The tradeoff between the amount of centrally controlled individuals and the price of anarchy is studied in [39].

Rosenthal [37] uses a discrete version of the potential function Φ to show existence of pure Nash equilibria in discrete congestion games. Fabrikant *et al.* [17] consider the complexity of computing Nash equilibria in a discrete model. They show that computing Nash equilibria is PLS-complete in general whereas there exists a polynomial time algorithm for single-commodity network congestion games. The first is reproved in [1], which also gives a nice characterisation of congestion games in which the sequential best response dynamics converges quickly.

One way to compute Wardrop equilibria is by convex optimisation. Various heuristics like the distributed algorithm presented by Bertsekas and Tsitsiklis [8] for the non-linear multi-commodity flow problem proceed by shifting certain amounts of flow from one path to another. Since, like in our case, these algorithm face the problem of overshooting, the amount of flow shifted depends in a linear fashion on the reciprocal of the second derivative of the latency functions. Recently, Awerbuch *et al.* [3] presented an efficient distributed algorithm for multi-commodity flow problems restricted linear latency functions.

The approach of no-regret routing models a dynamic population from the perspective of on-line learning. The *regret* is defined as the difference between an agent's average latency over time and the latency of the best path in hindsight (see, e. g., [4,9]). The goal is to minimise the regret over time. The bounds obtained here also depend in a pseudopolynomial fashion on network parameters.

Going to discrete models, analyses become more involved since probabilistic effects have to be considered. Consequently, results obtained for discrete models are more restrictive in other aspects, e. g., by limiting the strategy space to parallel links. Even-Dar and Mansour [16] study distributed and concurrent rerouting policies in such a model. They consider parallel links with speeds. Upper bounds are presented for the case of agents with identical weights. Their algorithms use static sampling rules that explicitly take into account the speeds of the individual links. Berenbrink *et al.* [7] present an efficient distributed protocol for balancing identical jobs on identical machines.

Traffic Engineering. Many works have been published on traffic engineering, i. e., the process of optimising the routing in a network, and related areas [24,48,25,46] such as traffic matrix estimation [38,49,45,50]. Most of the TE methods frequently used within the network operator community are offline methods: Network traffic intensities are measured over some period of time (usually a few hours); then these measurements are collected at a central point, where a new routing (e. g., readjusted OSPF weights; [25,26,5] and many others) is calculated. This new routing is then applied to the network, and the network remains in the new state — unless a link or a router fails, or the TE is re-run.

Such TE schemes achieve good performance gains. By running TE mechanisms multiple times per week or even per day, Internet providers can react to long-term changes in traffic demands. Due to a number of reasons, this process cannot be run more frequently, since this would affect the stability of the routing and cause a number of problems [29]. However, offline methods that are run once every few hours cannot react to sudden and unpredictable traffic changes in real time. Such changes can, e. g., be caused by BGP reroutes, flash crowds, malicious behaviour, and to some extent by diurnal traffic variations. Furthermore, while current TE can deal with network failures by pre-computing alternative routings, it usually does so only for a limited set of failures. It thus may fail to anticipate certain failures, which can lead to bad performance even when the network actually provides enough capacity.

In the past, various methods for online traffic engineering have been proposed. The earliest approaches to such load-adaptive routing were used in the Arpanet, but showed heavy oscillations due to interactions between the decisions made by the various routers [30]. Due to these bad experiences, routing in the Internet was designed not to react to traffic conditions, but only to topology changes. Instead, the mechanisms to protect the network from overload were moved from the routing protocol into the end hosts, mainly TCP congestion control. These issues in mind, network operators are sceptical about employing load-adaptive routing protocols. Instead, they confine themselves to traditional offline TE techniques, combined with vast overprovisioning of their network backbone infrastructure.

Their reluctance to employ load-adaptive routing is amplified by the fact that most load-adaptive schemes that have been proposed in theory require them to entirely replace their current routing architecture with a new load-adaptive one. To remedy this aspect, approaches that allow automatic online traffic engineering on top of an existing traditional routing infrastructure have been proposed; for an overview, we refer the reader to our work [29]. One of these approaches is our ReplEx algorithm presented in Section 5.

2 Evolutionary Selfish Flow

We envision the flow f to be controlled by an infinite population of agents, so f_P is both the flow on path P and the fraction of agents utilising path P. We start by describing a simple improvement process for the agents and evaluating it in the fluid limit, i. e., letting the number of agents go to infinity. This process will result in a system of differential equations specifying the behaviour of the flow, or population, f over time. We assume that agents become active at certain Poisson rates. Consider an agent in commodity $i \in [k]$ that is activated and currently uses path $P \in \mathcal{P}_i$. It performs two steps:

1. *Sampling:* Sample another agent uniformly at random. This agent uses path $Q \in \mathcal{P}_i$ with probability f_Q/r_i.
2. *Migration:* Migrate from path P to path Q if $\ell_P(f) > \ell_Q(f)$ with probability proportional to the latency improvement $\ell_P(f) - \ell_Q(f)$.

Given these probabilities and summing up inflow and outflow over all paths we obtain in the fluid limit the following dynamic system:

$$\dot{f}_P = \lambda_i(f) \cdot f_P \cdot (L_i(f) - \ell_P(f)) \quad \text{for } i \in [k],\ P \in \mathcal{P}_i \qquad f(0) = f_0. \tag{2}$$

Here, the constant factor λ_i accounts for the constant of proportionality in step 2 and the Poisson rate at which the agents are activated.

We believe that this is a reasonable model of communication and selfish behaviour for several reasons. The steps involved in the process are simple, based on local information, and are stateless. Furthermore this behaviour is compatible with selfish incentives of the agents. It does neither depend on accurate knowledge of the network and the latency functions nor on the assumptions of unbounded rationality and reasoning capabilities. Note, however, that our population model still assumes perfect information in that actions made by other agents become visible to the other agents immediately. Subsequently we will see how we can abandon this assumption as well.

In order to show convergence towards Wardrop equilibria, let us consider the behaviour of the potential Φ over time. The time derivative is $\dot{\Phi}(f) = \sum_{e\in E} \dot{f}_e \cdot \ell_e(f_e)$. Substituting the replicator dynamics (2) into this it is easy to see that

$$\dot{\Phi}(f) = \sum_{i\in[k]} \lambda_i(f) \cdot r_i \cdot \left(L_i(f)^2 - \sum_{P\in P_i} \frac{f_P}{r_i} (\ell_P(f))^2 \right) \ . \tag{3}$$

By Jensen's inequality, each term of this sum is non-positive and negative if f is not a Wardrop equilibrium. Using Lyapunov's direct method, this yields:

Theorem 1 ([21]). *For $i \in [k]$, let $\lambda_i(\cdot)$ be a continuous, positive function. Let $\xi(f_0, t)$ be a solution to the replicator dynamics (2) for some initial population with $f_P(0) > 0$ for all $P \in \mathcal{P}$. Then, $\xi(f_0, t)_{t\to\infty} \to \mathcal{F}_{\Gamma}^{NE}$.*

3 Stale Information

The convergence result presented in the preceding section shows that, in the long run, a population of selfish agents is able to attain a Wardrop equilibrium in a distributed way. However, our dynamic population model treats the aspect of concurrency in an idealistic fashion. It is implicitly assumed that any effect that the migration of population shares has on latency can be observed without a delay. In this sense, we have still assumed perfect information. A reasonable model of concurrency, however, should take into account the possibility that information may be stale: In practise, e. g., a rerouted flow may unfold its full volume only after a re-initialisation phase, and “observing” the latency of a path may actually comprise measurements taken over a certain period of time. In such a scenario, naive policies like the best response dynamics may lead to overshooting and oscillation effects. These effects are well-known in practise and may seriously harm performance [31,32,36,43]. In fact, oscillation effects are a major obstacle in designing real-time dynamic traffic engineering protocols. Similar negative results have been obtained in theory [10,34].

The strength of these oscillation effects chiefly depends on three quantities: the maximum age of the information, the sensitivity of the latency functions to small changes of the flow, and the speed at which the rerouting policy shifts flow. Whereas the first two parameters are determined by the system, the third is controlled by the rerouting policy, e. g., in case of the replicator dynamics, by the scalar function $\lambda_i(f)$ in Equation (2).

In [22], we ask the question of how agents should behave in order to guarantee convergence to Wardrop equilibria despite the fact that information may be stale. As a model of stale information we consider the bulletin board model introduced by Mitzenmacher [34] in the context of dynamic load balancing. In this model, all information relevant to the routing process is posted on a bulletin board and updated at regular intervals of length T. The information on the bulletin board is accessible to all agents. Again, agents may reroute their traffic at points in time that are the arrival dates of a Poisson process. We will see that in the bulletin board model, naive rerouting policies like the best response policy oscillate and fail to converge and identify a class of policies for which convergence is guaranteed.

3.1 Smooth Rerouting Policies and Stale Information

Smooth Rerouting Policies. We consider a very broad class of adaptive routing policies that can be described in the following way. Each agent becomes active at points in time generated at fixed Poisson rates for each agent independently and performs the following two steps. Consider an agent of commodity $i \in [k]$ currently using path $P \in \mathcal{P}_i$.

1. *Sampling:* The agent samples a path $Q \in \mathcal{P}_i$ according to some probability distribution. Let $\sigma_{PQ}(f)$ denote the probability that an agent currently using path P samples path Q.
2. *Migration:* The agent switches to the sampled path Q with probability $\mu(\ell_P, \ell_Q)$.

We illustrate the class of policies in this framework by giving some examples for possible sampling and migration rules.

Typically, $\sigma_{PQ}(f)$ will be independent of the originating path P. The simplest sampling rule is the *uniform sampling* rule which assigns uniform probabilities to all paths, i. e., $\sigma_{PQ}(f) = 1/|\mathcal{P}_i|$ for all $i \in [k]$ and $P, Q \in \mathcal{P}_i$. For the replicator dynamics, we have applied a *proportional sampling* rule where $\sigma_{PQ}(f) = f_Q/r_i$. Note that a population cannot converge towards a Wardrop equilibrium if $\sigma_{PQ}(f)$ may be zero for some path Q that is used in every Wardrop equilibrium. Hence, we also require $\sigma_{PQ}(f) > 0$ for all $P, Q \in \mathcal{P}$ and every f that can be reached from an initial state f_0. Note that the proportional sampling rule satisfies this condition if $f_0 \in \text{int}(\mathcal{F}_\Gamma)$.

Let us now give two examples for the migration rule μ. A natural, but, as we will see, deceptive, migration rule is the *better response policy* defined by $\mu(\ell_P, \ell_Q) = 1$ if $\ell_Q < \ell_P$ and $\mu(\ell_P, \ell_Q) = 0$ otherwise. Another natural migration rule is to switch from P to Q with a probability proportional to the latency difference: $\mu(\ell_P, \ell_Q) = \max\{0, \frac{\ell_P - \ell_Q}{\ell_{\max}}\}$ where $\ell_{\max}$ is the maximum latency of a path. This corresponds to the migration rule of the replicator dynamics. We call this migration rule *linear migration policy.* We assume that $\mu(\ell_P, \ell_Q) > 0$ if and only if $\ell_Q > \ell_P$ and μ is non-decreasing in ℓ_P and non-increasing in ℓ_Q.

As discussed above we need to assume that the migration rule is not too greedy. We formalise this requirement in the following way:

Definition 2 (α-smoothness). *We say that a migration-rule $\mu : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \mapsto [0, 1]$ is α-*smooth *for some $\alpha > 0$ if for all $\ell_P \geq \ell_Q \geq 0$ the migration probability satisfies $\mu(\ell_P, \ell_Q) \leq \alpha \cdot (\ell_P - \ell_Q)$.*

We can now describe the evolution of the population shares over time in terms of the fluid limit. Given the sampling and migration rule and normalising the Poisson activation rate of the agents to be 1, we can compute the rate $\rho_{PQ}(f)$ at which agents migrate from path P to path Q: $\rho_{PQ}(f) = f_P \cdot \sigma_{PQ}(f) \cdot \mu(\ell_P(f), \ell_Q(f))$. Summing up over all paths Q we obtain a system of ordinary differential equations describing the growth rates (or derivatives with respect to time) of the population share utilising path P:

$$\dot{f}_P(t) = \sum_{Q \in \mathcal{P}_i} (\rho_{QP}(f(t)) - \rho_{PQ}(f(t))) \qquad f(0) = f_0 \ . \tag{4}$$

Stale Information. The model of stale information we use in this section was originally introduced by Mitzenmacher [34]. In this model, agents do not observe latency and flow values in the network directly, but rather consult a bulletin board on which this information is posted. The bulletin board is updated only infrequently, at intervals of length T. Using the information on the bulletin board for the evaluation of the sampling and migration rule, our expression for $\dot{f}(t)$ now depends not only on $f(t)$ but also on $f(\hat{t})$ where $\hat{t}$ marks the beginning of the respective update period. Fix an update period length T and let $\hat{t} = \lfloor t/T \rfloor \cdot T$ denote the beginning of a phase containing t. Then, the migration rate becomes

$$\hat{\rho}_{PQ}(f(t)) = f_P \cdot \sigma_{PQ}(f(\hat{t})) \cdot \mu(\ell_P(f(\hat{t})), \ell_Q(f(\hat{t})))$$

and our dynamic system becomes

$$\dot{f}_P(t) = \sum_{Q \in \mathcal{P}_i} (\hat{\rho}_{QP}(f(t)) - \hat{\rho}_{PQ}(f(t))) \qquad f(0) = f_0 \ . \tag{5}$$

Stale vs. Recent Information. Assuming up-to-date information, convergence towards Wardrop equilibria can easily be shown using the Beckmann-McGuire-Winsten potential [6] Φ as defined in Equation (1). Under stale information, however, convergence is not guaranteed. Let us consider the best response dynamics in which every agent whenever activated migrates to a path with lowest latency. There exists a simple instance with two parallel links and latency functions with maximum slope s such that the best response policy periodically generates a flow in which more than half of the agents sustain a latency of $\Omega(T\,s)$ whereas the minimum latency is 0.

3.2 Convergence under Stale Information

We have seen that under up-to-date information the potential decreases. More precisely, if agents migrate from path P to path Q at rate ρ_{PQ}, they cause an infinitesimal potential gain of $\rho_{PQ} \cdot (\ell_Q - \ell_P)$ which is negative for selfish policies. Consider a phase in the bulletin board model starting with a population $\hat{f} \in \mathcal{F}_\Gamma$ and ending with a population $f \in \mathcal{F}_\Gamma$. Let Δf_{PQ} denote the fraction of agents migrating from path P to path Q within this phase. Since throughout the phase the latency values on the bulletin board are fixed to the values they had at the beginning of the phase, agents "see" a *virtual potential gain* of $V_{PQ} = \Delta f_{PQ} \cdot (\ell_Q - \ell_P)$.

Unfortunately, summing up over all pairs of paths, this does not yield the true potential gain since the latency values do actually change during the phase. The *error terms* $U_e = \int_{\hat{f}_e}^{f_e} (\ell_e(u) - \ell_e(\hat{f}_e))\, du$ account for this difference. It is easy to check that the virtual potential gain terms and the error terms sum up to the true potential gain.

For the class of policies under consideration, every virtual potential gain term V_{PQ} is non-positive and at least one is negative. We can show that if the rerouting policy is smooth enough, the real potential gain is still at least half of the virtual potential gain and is thus, in particular, negative. The condition on the smoothness depends linearly on the maximum slope, the update period length T, and the maximum path length D. We can prove this fact using the following observation: In order for an error term to be large, a large volume of agents must migrate towards the respective edge. This volume is only large, however, if these agents also see a large latency gain. Hence, a large virtual potential gain is made. Based on this fact, convergence can be shown:

Theorem 2. *Suppose that the migration rule $\mu(\cdot,\cdot)$ is α-smooth, the slope of the latency functions is bounded from above by s, and the bulletin board is updated at intervals of length $T \leq 1/(4\,D\,\alpha\,s)$. Then, the solution $\xi(f_0, t)$ of the dynamical system given by Equation (5) in the bulletin board model converges towards the set of Wardrop equilibria, i. e., $\xi(f_0, t)_{t\to\infty} \to \mathcal{F}_\Gamma^{NE}$.*

Our protocol requires that the agents are not too greedy. Let us remark that for an individual player it may still be advantageous to deviate from the protocol. In this sense, our protocol still relies on the cooperation of the agents.

3.3 Fast Convergence

The analysis in the preceding section is not yet satisfactory with respect to the speed of convergence. By requiring α-smoothness, we have effectively slowed down the dynamics by a factor that is proportional to the maximum slope of the latency functions. Unfortunately, this quantity is unbounded for some natural classes of latency functions, e. g., for polynomials of fixed degree, as well as for any class that is closed under scaling. Even for any fixed instance the maximum slope may be large. The goal of this section is to derive bounds on the time of convergence that are polynomial in the representation length of Γ, e. g., if latency functions are represented as polynomials in coefficient representation. Thus, the bounds are much stronger than the ones that can be obtained for policies presented in the preceding section. To that end, we analyse a very specific rerouting policy, rather than the large class of smooth adaption policies considered in Section 3.2.

Again, we study policies consisting of a sampling step and a migration step. Our policy applies a proportional sampling rule in order to exploit the exponential growth rate of good strategies and, with small probability, applies uniform sampling in order to explore the strategy space. This time, we use the maximum elasticity[1] d of the latency functions as a damping factor, rather than the slope. The elasticity is a parameter which behaves much nicer than the maximum slope. E. g., for positive polynomials of degree d, the elasticity is at most d.

Our exploration-replication policy uses a mixture of proportional and uniform sampling, where the ratio is controlled by the parameter β. In every round, an agent is activated with constant probability λ/d where λ is a suitable constant. It then performs the following two steps. Consider an agent in commodity $i \in [k]$ currently utilising path $P \in \mathcal{P}_i$.

[1] The elasticity of a function ℓ at x is $\frac{x\cdot\ell'(x)}{\ell(x)}$. For example, the elasticity of $\ell(x) = a\cdot x^d$ is d.

1. *Sampling:* With probability $(1-\beta)$ perform step 1(a) and with probability β perform step 1(b).
   (a) *Proportional sampling:* Sample a path $Q \in \mathcal{P}_i$ with probability f_Q/r_i.
   (b) *Uniform sampling:* Sample a path $Q \in \mathcal{P}_i$ with probability $1/|\mathcal{P}_i|$.
2. *Migration:* If $\ell_Q < \ell_P$, migrate to path Q with probability $\frac{\ell_P - \ell_Q}{\ell_P}$.

Note that the migration probability is now proportional to the *relative* improvement and is hence much more aggressive than the standard linear migration rule. Although not satisfying the smoothness condition from the preceding section, we can still show that our dynamics converges towards the set of Wardrop equilibria as in Theorem 2, if the amount of exploration β is chosen small enough.

To analyse the time of convergence we consider two kinds of approximate equilibria. For our first result, we may even set $\beta = 0$. We consider a bicriterial definition of approximate equilibria which allows for a deviation from L_i that is relative to L. More precisely, we require all but an ϵ-fraction of the agents to deviate by no more than δL in both directions from the average latency of their commodity. Such a flow is called a *(δ,ϵ)-equilibrium* . In the single-commodity case, this can be interpreted as an allocation where almost all agents are almost satisfied. In [20] it is shown that the time to reach an approximate equilibrium in this relative sense is bounded from above by

$$\mathcal{O}\left(\frac{d}{\delta^2\,\epsilon^2}\log\left(\frac{\Phi(f_0)}{\Phi^*}\right)\right)$$

where d is the elasticity of the latency functions. Remarkably, this bound is almost independent of the size of the network. This bound also holds when $\beta = 0$, i. e., uniform sampling is deactivated.

Let us briefly sketch the proof of this theorem for the single commodity case. Consider a population that is not yet at a (δ,ϵ)-equilibrium. Then, there exists an ϵ-fraction of agents that that can improve by $\delta\, L$ by migrating from their path to an average latency path. If such a path is sampled, the migration probability is δ. Since $\Theta(1/d)$ agents are activated per round, this results in a potential gain of $(\epsilon\,\delta^2/d)\,L \geq (\epsilon\,\delta^2/d)\,\Phi$. Hence, we have that $\Phi(t+1) \leq \left(1 - \Omega\left(\epsilon\,\delta^2/d\right)\right)\Phi(t)$. Using Φ^* as a lower bound for $\Phi(t)$, this yields the desired result.

Bicriterial approximate equilibria in this sense are transient: A minority of agents utilising low latency paths may grow to a volume of more than $\epsilon \cdot r$ at a later time, and it may take an arbitrarily long time until this event happens if the initial flow on these paths is arbitrarily small.

For single-commodity instances, this can be resolved by considering approximate equilibria in terms of the potential Φ and activating uniform sampling, i. e., setting $\beta > 0$. Thus, we obtain a non-zero lower bound on the flow of any path with below average latency after a single round already, which can be boosted by the proportional sampling rule afterwards. Note that $1/\beta$ must be chosen pseudopolynomially large. Still, due to the proportional sampling rule, our time bounds are polynomial. More precisely, the time to reach a flow with potential at most $(1+\epsilon)\,\Phi^*$ is bounded from above by

$$\operatorname{poly}\left(\frac{d \cdot D}{\epsilon}\right) \cdot \operatorname{polylog}\left(|E| \cdot \frac{\ell'_{\max}}{\ell_{\min}}\right) \cdot \log\left(\frac{\Phi(f_0)}{\Phi^*}\right)$$

where $\ell'_{\max}$ is an upper bound on the maximum slope of the latency functions and $\ell_{\min}$ is the minimum edge latency. This bound is polynomial in the representation length of the instance Γ if, e. g., the latency functions are positive polynomials of arbitrary degree in coefficient representation. Thus, the exploration-replication policy behaves like a fully polynomial time approximation scheme.

The two take home message of this section are that first, proportional sampling can significantly improve convergence time and second, the critical parameter that limits the convergence speed is the elasticity of the latency functions.

4 Computing Approximate Equilibria

Based on the above results for infinite populations we can design a distributed algorithm to compute approximate Wardrop equilibria. Unfortunately, the exploration-replication policy does not immediately yield an algorithm since it can handle an exponential number of paths only due to the fact that the number of agents is infinite. In [19] we present an algorithm that solves this problem using a randomised approach. For every commodity we use an agent that strives to balance the latency of its commodity. Hence, the overall computed flow will be at a Wardrop equilibrium.

To compute flow updates in polynomial time, our algorithm uses an edge flow vector $(f_e)_{e \in E}$ rather than an exponentially long path flow vector. From the edge flows, we infer a natural implicit path decomposition, denoted $\tilde{f}$. The amount of flow $\tilde{f}_P$ assigned to the set of paths containing edge $e = (v, w)$ is proportional to the share the edge e has in the flow through v. If we assume that the underlying graph is a directed acyclic graph (DAG), we can apply this rule in any topological ordering of the nodes and obtain $\tilde{f}_P = r_i \cdot \prod_{(v,w) \in P} f_{(v,w)}/f_v$.

This implicit flow decomposition $\tilde{f}$ is sampled to obtain another, randomized, decomposition $(f_P)_{P \in \mathcal{P}}$ using only few paths. To that end we sample a path P where the probability to sample a path is proportional to its flow in $\tilde{f}$. Now, P is assigned a certain flow volume f_P. For the time being, assume that we assign the entire bottleneck flow to P. (The bottleneck flow is the minimum edge flow along the path.) Then, if P has latency above L_i, we remove a portion of $x = \Theta\left(f_P \cdot (\ell_P - L_i)/(d\,\ell_P)\right)$ of its flow and distribute it proportionally among all admissible paths. As long as we are not at a δ-ϵ-equilibrium, the probability of sampling a path for which a large latency gain is possible is not too small. In addition, we can lower bound the probability that the bottleneck flow of the sampled path is not too small.

To increase the potential gain further and to concentrate it around its expectation, we repeat this process $T = m \log m$ times per round, each time consuming at most a $\Theta(1/\log m)$ fraction of the bottleneck flow. Then, the probability that an edge becomes empty within one round becomes very small. Pseudocode is

Algorithm 1. RandomisedBalancing(d) (executed by all commodities in parallel; $(f_e)_{e \in E}$ denotes the edge flow vector of commodity i)

```
1: for $T = m \log m$ times do
2:     sample a path $P$ where $\Pr P = \frac{\tilde{f}_P}{r_i}$
3:     let $b$ denote the bottleneck flow of $P$; let $f_P = \min\left\{\frac{1}{7\,m\,\log m}, \frac{b}{7\,\log m}\right\}$
4:     if $\ell_P > L_i$ then
5:         reduce the flow on all edges $e \in P$ by $\Delta f_P = f_P \cdot \frac{\ell_P - L_i}{4\,d\,\ell_P}$
6:         if for any $e \in P$, $f_e < 0$ then
7:             undo last action, abort loop, and continue in line 11
8:         end if
9:     end if
10: end for
11: increase the flow on all edges $e \in E$ proportionally by $\frac{f_e}{r_i} \cdot \sum_{P \in \mathcal{P}_i} \Delta f_P$
```

given in Algorithm 1. Combining the techniques from the previous sections with the arguments sketched above, we can prove the following theorem:

Theorem 3 ([19]). *Assume that $f_e(0) > 0$ for all $e \in E$. Then, the sequence of flow vectors computed by Algorithm* RandomisedBalancing *converges towards the set of Wardrop equilibria. Furthermore, the expected number of rounds in which the flow is not at a (δ,ϵ)-equilibrium is bounded by*

$$\mathcal{O}\left(\frac{d}{\epsilon^3\,\delta^2}\log\left(\frac{\Phi(f_0)}{\Phi^*}\right)\right) \quad ,$$

if d is an upper bound on the elasticity of the latency functions. The computation time of each round is bounded by $\mathcal{O}\left(n \log n \cdot m \log m\right)$.

5 Wardrop Routing in an IP Network

The Wardrop model draws on a rather theoretical setup: It is suitable when the number of agents is infinite, the agents have full control over their traffic and easy access to the current latency of the paths. However, none of these requirements is true in real-world networks such as the Internet. In the following we address these issues and show that a variant of our Wardrop policy can yield a practical algorithm that can be deployed in practice.

The first major problem is that agents normally do not have control over the path their packets take in real-world communication networks — usually, the end points of a communication stream are located at the periphery of the network, and are typically connected to the network by only a single link. Rather, it is the routers that are responsible for choosing the path along which the data packets are sent. Moreover, in most cases not even the routers can choose a path as a whole: normally, a router can only determine the next-hop node on the way to the destination routing is used. In normal IP routing this decision is solely based on the destination of a packet while other header attributes (e. g., the source) are neglected. This has the effect that not all possible paths through the network actually can be used. Hence the mapping of Wardrop agents to a network is non-obvious.

We solve this problem as follows: In order to control the traffic balance, every router r maintains a set of dynamically changeable weights $w(r, t, v_i)$ for every target t and possible next-hop neighbour v_i. We can interpret $w(r, t, v_i)$ as the exact fraction of traffic routed from r to t via v_i (i. e., $r \to v_i \rightsquigarrow t$). By adjusting their weights $w(\cdot)$, the (finite) number of routers along all possible paths from source s to sink t *jointly* perform the calculations for the (infinite) number of agents residing at s from the Wardrop model. In other words, the decisions of one agent from the theoretical model are implemented by a chain of multiple routers, whereas one router performs the (partial) decisions of (infinitely) many agents. By adhering to the Wardrop model and the exploration-replication strategy, we have the theoretical guarantee for quick convergence.

From the perspective of a router r, in order to make the routing decision, r only needs traffic information about the set of paths between r and t. Also note that r cannot control which one of the possible paths a packet will take once it has been forwarded to one of the v_i's. Thus, r cannot exploit exhaustive information about *all* possible paths. On the other hand, r requires to know more information on the network conditions than it can obtain from local measurements; otherwise bottleneck links further downstream would not be visible to r. Our solution is thus to use only aggregated information about the possible paths from next-hop router v_i to destination t: Each router monitors the congestion condition on its outgoing interfaces, and regularly exchanges information on network-wide traffic conditions with its neighbours using a distance-vector like protocol.

Another issue we need to address in a real network is how we actually perform the traffic split that the weights prescribe. The naïve way of distributing traffic according to some weights $w(r, v_i, t)$ would be to use either some round-robin fashion or simple randomisation, where each weight is interpreted as a probability for routing packets along the corresponding path section. However, this probabilistic routing approach has the drawback that packets headed towards the same destination may take different paths. If this happens among packets pertaining to the same TCP connection, they can overtake each other, which seriously harms TCP performance [33]. Since most traffic in the Internet is TCP traffic [15], this is inacceptable.

Therefore we "roll the dice" not per-packet but rather per-flow by applying the well-known technique [11] of using a hash function: For each packet, we calculate a hash value based on its source and destination address, and use this pseudorandom value for our weight-based probabilistic decision. Naturally, packets pertaining to the same TCP stream feature the same header data and thus always hash to the same values; hence they are always treated in an identic fashion.

So far we have assumed that every router r is able to measure ℓ for all outgoing edges $\{r, v_i\}$. With an acceptable computational effort we can measure the following values during each time interval: the number of packets sent and dropped (due to link overload), the number of bytes sent and dropped, and the average queue length. From this information we can compute a number of interesting metrics, depending on our optimisation objective [29,18], e. g., link load or cumulative queueing delays. In particular, we can calculate the *packet loss probability*, short *loss rate*. We compute this probability p as the number of dropped packets per interface, divided by the number of packets that were meant to be sent out via that interface. The measurements of the different routers along a path taken together, we can calculate the overall packet loss probability along the entire path, and use this figure as an indicator for the path's service quality.

Another challenge is the very bursty [35] nature of Internet traffic. Therefore, traffic measurements made within a short time interval are not very reliable. To this end, instead of using the measured values directly, we use an exponential moving average (EMA).

5.1 The ReplEx Algorithm

Combining the techniques presented in the previous section, we now present the protocol which is run on each individual router that participates in a network optimised by our algorithm. Since it is derived from the exploration-replication policy, we name our algorithm ReplEx.[2] Recall that a protocol implementation has to address several tasks: Foremost, it needs to calculate the weights $w(\cdot)$. For this, measurements $\ell(\cdot)$ on the router's link queues are necessary. These are combined with the information $A(\cdot)$ that is received from neighbouring routers via our distance-vector like information protocol. Finally, it has to compute the information $L(\cdot)$ which it then sends to its neighbours. In order to perform its computations, each router r maintains the following arrays:

$\tilde{\ell}(r,v)$: The EMA of the measurements for link (r,v).

$L(r,t,v)$: metric value for destination t using next hop node v

$A(v,t)$: The average measurement value that next hop router v has announced for destination t.

$w(r,t,v)$: Current weight of route $r \rightarrow v \rightsquigarrow t$ (i. e., route for destination t via next hop neighbour v)

In the router's main loop, which is executed every T seconds, these values are updated and sent to the neighbouring routers. Algorithm 2 describes the procedure in more detail. At the heart of the algorithm we have the actual weight adaption procedure described in Algorithm 3. For every pair of next hop nodes, this procedure computes the migration rates (line 5). The computed rates are then migrated by shifting the corresponding weights.

Obviously, the ReplEx algorithm depends on a number of different parameters, which have been introduced to in the previous subsections. In the following we give an overview on the static parameters of our algorithm.

update period length T: The length of the interval at which the main loop is executed.

[2] . . . Which happens to sound better than ExRep.

Algorithm 2. Main loop of the ReplEx algorithm

```
initialise an empty message $M$
for each destination $t$ in the routing table do
    for all next-hop nodes $v \in N(r,t)$ do
        measure performance $\ell_{rv}$ and update moving average $\tilde{\ell}(r,v)$
        $L(r,t,v) \leftarrow \mathrm{agg}(\ell_{rv}, A(v,t))$.
    end for
    $avg \leftarrow \sum_{v \in N(r,t)} w(r,t,v) \cdot L(r,t,v)$
    Append $(t, avg)$ to $M$
end for
send $M$ to all neighbours $v$ who store it in $A(r,\cdot)$
call procedure AdaptWeights (Algorithm 3)
```

Algorithm 3. Procedure AdaptWeights

```
for each destination $t$ in the routing table do
    $w'(r,t) \leftarrow w(r,t)$.
    for each pair of next-hop routers $v_1, v_2 \in N(r,t)$ do
        if $L(r,t,v_1) > L(r,t,v_2) + \epsilon$ then
            $\delta \leftarrow \lambda \left( (1-\beta) w(r,t,i) + \frac{\beta}{|N(r,t)|} \right) \frac{L(r,t,v_1) - L(r,t,v_2)}{L(r,t,v_1) + \alpha}$
            $w'(r,t,v_1) \leftarrow w'(r,t,v_1) - \delta$
            $w'(r,t,v_2) \leftarrow w'(r,t,v_2) + \delta$
        end if
    end for
    set $w(r,t) \leftarrow w'(r,t)$.
end for
```

weight shift factor λ: This parameter determines the weight shifted in one round. The ratio λ/T controls the convergence speed of the algorithm. Oscillation effects and congestion control feedback loops of affected TCP connections limit the maximum convergence speed we can achieve.

virtual latency offset α: This virtual offset is added to all path latencies $L(r,t,v)$ making the algorithm less sensitive to small differences when the metric is close to 0.

improvement threshold ϵ: The parameter ϵ can be considered a damping factor. Projected weight shifts that are smaller than ϵ are not carried out; thus we can ensure that weights are only shifted if the change is substantial.

exploration ratio β: This parameter determines the ratio between replication and exploration. If β is too small, currently unused paths may not be detected by our algorithm. On the other hand, β should not be too large since this can result in excessive growth of a flow if f_P is close to zero.

metric to optimise: As we pointed out before, ReplEx is not restricted to operate with latencies. Rather, the ReplEx instances in a network also can be used to optimise other network performance metrics, e.g., maximising throughput, minimising link loads, or minimising packet losses.

Our simulations show that the performance is largely independent of the choice of α, β and ϵ, as long as these are chosen within reasonable intervals. The more sensitive parameters that influence the performance most are T, λ, and the metric that the individual ReplEx instances aim to optimise. Extensive simulation-based analyses of the parameter space are given in [29,18].

6 Evaluation of the ReplEx Algorithm

We now show some of our evaluation results for the ReplEx algorithm using network simulations. It is our intention to show that our protocol behaves well with realistic network traffic. Therefore, we employ traffic loads that mimic actual Web traffic, i. e., bursty TCP traffic whose characteristics are consistent with self-similarity. We simulate ReplEx in complex realistic network topologies, i. e., the EBONE (AS 1755), Exodus (AS 3967), Metromedia Fiber / Abovenet (AS 6461) network topologies from the 2002 Rocketfuel [44] data, and the AT&T topology which is shipped as an example together with the TOTEM toolbox [5]. We first simulate the network with ReplEx agents disabled for some time. Then, at a specific point in time labelled as $t = 0$, we enable ReplEx on all routers, and examine the ensuing developments in the network for $t > 0$. Further details on our simulation setup can be found in [29].

Based on our observations in [29] and [18], we choose $\lambda = 0.5$, $T = 1\,\mathrm{s}$, $\mathit{metric} =$ drop rate, $\alpha = 0.25$, $\beta = 0.1$, $\epsilon = 0.0$ as parameters for ReplEx. The time interval implies that one pair of ReplEx messages is exchanged per second between every pair of neighbouring routers. Assuming that each ReplEx message contains a 6-byte information for 2,000 destination prefixes, this amounts to a total of just 96 kbit/s, i. e., just 0.32 % of additional traffic on each 30 Mbit/s link — the additional signalling overhead for ReplEx is negligibly small.

6.1 Measuring Performance

We examine the performance under two criteria: First, how much does applying ReplEx increase the "performance" of the network; and second, how fast does ReplEx achieve this goal? This opens two questions: how do we define the "performance" of the network, and how do we define the "speed" of convergence, i. e., how do we define "convergence"?

Network Performance

A number of criteria can be thought of as performance measures. Since a common optimisation goal in traditional (i. e., offline) traffic engineering is to minimise the maximum link load on any link in the network, this measure is naturally an interesting performance indicator.

One of the reasons to use this as a goal in traditional TE is that a high load on a link increases the probability for packets to be dropped when they are to be sent over it. An increased loss rate, however, does not only decrease the quality of service as perceived by the end users — it can, moreover, lead to synchronisation effects across otherwise independent TCP connections, since all connections affected by the same packet loss burst will react with their congestion control mechanisms (slow start or fast retransmit) around the same time. This synchronisation, in combination with the effectively nonlinear increase of the slow-start phase, will increase the traffic's burstiness and thereby deteriorate the service quality even more. Therefore, the packet loss rate is another useful measure for network performance.

Another measure that is of interest for the network operator is the *network throughput*, i. e., number of bytes that are transmitted through the network, since financial revenues generally increase with the volume of transmitted data. On the other hand, we can assume that TCP congestion control will make use of additional network capacities, thereby shortening download times and thus allowing more downloads per time unit.

Further useful performance measures are described and analysed in [29].

Determining Convergence

Since ReplEx is a dynamic traffic engineering algorithm, it is also interesting to examine how *fast* ReplEx can attain an improved performance level, i. e., we need to measure its speed of convergence.

In [29] we find that some of our performance measures seem to reach their optimal level quicker than others — for example, as soon as we enable ReplEx, the number of transmitted bytes increases much faster than the route weights change. Therefore, although it may at first sight feel natural to define convergence solely on the output of the algorithm, e. g., the changes to the weights, it is also interesting to see how quickly ReplEx stabilises at a level with increased network performance.

Since we use a statistical workload model, we have significant fluctuations in our network traffic even if we do not employ ReplEx at all. Therefore, we cannot expect any of our performance measures to monotonically increase over time and finally converge to one fixed value. Rather, it is natural for our performance measures to fluctuate — even during the convergence process. Our solution to this problem is to calculate the *statistical convergence* of the time series in question, after the method we describe in [29].

Measuring Weight Changes

The immediate output of our algorithm contains the weights and weight changes. In small and primitive test scenarios that feature only extremely few equal-cost routes, we could measure convergence times for each individual route. However, this approach is not feasible for the large and complex scenarios we intend to use. Instead, we sum up all weight changes that occur within one time interval. At the start of ReplEx, we expect the weight changes to occur with a large amplitude: At this point in time, the weights are completely un-tuned towards the actual traffic demands in the network, and ReplEx is constructed such that it applies larger weight changes if more gain is to be expected, whereas it applies only small changes if less gain is to be expected. The latter should be the case after some time has elapsed and the first weight shifts already have led to a significant increase in network performance.

Note that packet reordering, which negatively affects TCP performance, can still occur, albeit rarely, whenever ReplEx sends an active TCP connection along a different path. The probability for this to happen is linear with the weight change that ReplEx applies, so it is desirable to only make small weight changes. Thus, it makes sense to "punish" heavy increases or decreases in weight. Therefore we do not simply sum the weight changes per time unit, but sum up the *squared weight changes* per time unit as a means to capture ReplEx's activities.

6.2 Effectivity of ReplEx

We start with analysing the weight change behaviour of our algorithm in the topology AS 1755, which is described in [29] in greater detail. Due to the size of the topology, it is not feasible to show weight changes for individual routes or individual ReplEx instances. Instead, we only consider the total weight changes for each 1 second interval as described above. We expect the weights to be subject to small fluctuations even in the converged state, due to the bursty nature of the generated realistic workload traffic.

Fig. 1 shows the temporal development for the weight change metric. We see a sharp decline during the first ≈ 150 s. A converged state by our definition from [29] is reached roughly at $t = 400$ s (dashed vertical line).

To demonstrate ReplEx's performance improvements, Fig. 2 shows the overall network throughput, i. e., the total number of IP bytes received by all simulated clients per 1 second time interval. We see that the network performance increases slightly faster than the weight changes decrease within a time window of roughly 200 s (note that $t = 0$, the start of ReplEx actions, lies in the middle of the plot). Note that, in contrast to the weight changes, it does not make sense to apply our convergence definition, since here the range of our

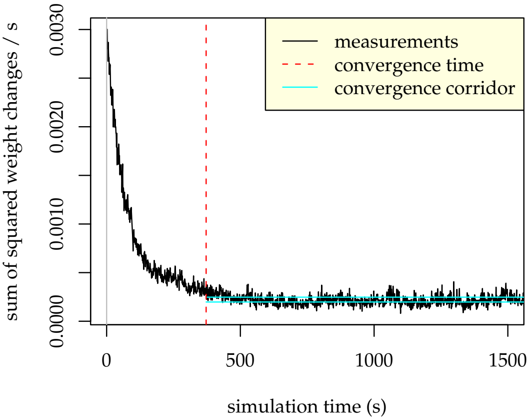


Fig. 1. Weight changes in AS 1755

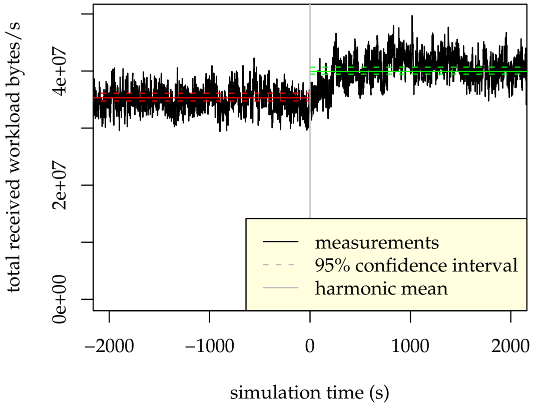


Fig. 2. Influence of ReplEx on network throughput in AS 1755

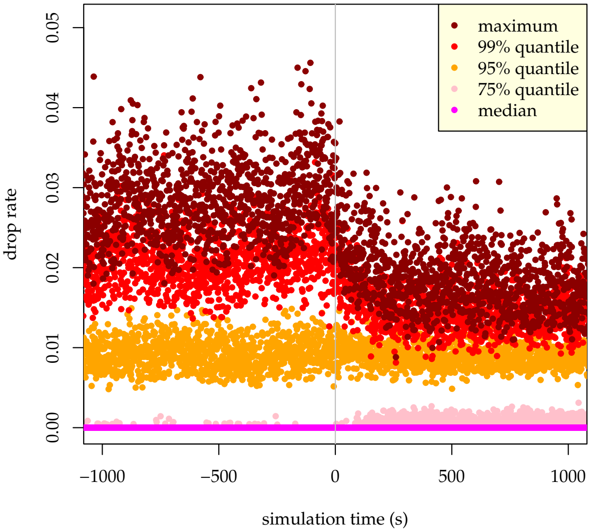


Fig. 3. Influence of ReplEx on the drop rates in AS 1755

measurements (i. e., throughput values) prior to and after starting ReplEx do overlap. Thus, we cannot be sure that reaching the convergence corridor is a sign of convergence, or just an outlier caused by a random fluctuation in the workload that might as well have happened without ReplEx. However, we can see that the throughput gain achieved by ReplEx is significant, since the 95 % confidence intervals[3] are clearly separated (dashed red and green lines).

Where do these gains in throughput come from? Naturally, the reason is a significant reduction in the number of packet drops. Fig. 3 shows the maximum

[3] Calculated on the means of 32 groups of 125 values.

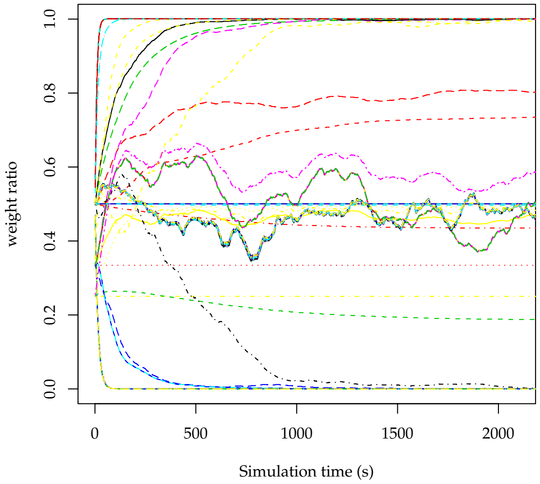


Fig. 4. Weight changes at randomly chosen router (AT&T topology)

value (brown) of the drop rates in the network, as well as their 99 % quantile (red), the 95 % quantile (orange), the 75 % quantile (pink) and the 50 % quantile (i. e., median, flat magenta-coloured line at bottom). We see a swift decrease in the maximum number and 99 % quantile of packet drops during the first ≈ 250 s after ReplEx has started. The 95 % quantile remains more or less unchanged, while the 75 % quantile slightly increases. The median of the link loads (and thus all lower quantiles as well) remains constantly at zero. In other words, a strong reduction of the most heavily congested links (brown, red) is achieved, at the cost of a slight increase in the loss rate on less congested links (pink).

The results of this and some other simulations from [29] are summarised in Table 1. We witness a performance increase of 13.1 % for ReplEx over the unoptimised network in the AS 1755 topology (first block, second row), of 8.5 % for the AS 3967 topology (second block, second row), and 6.5 % for AS 6461 (third block) through the use of ReplEx. Furthermore, we achieve a very good performance increase of 16.4 % for the AT&T topology (fourth block).

When we examine the weight changes of all routes at one randomly chosen router (Fig. 4), we see that indeed many weights converge within less than 100 s (left). In many cases, we observe weight settings where one route alternative is completely disabled (i. e., either weight = 0 or weight = 1). Other route weights, in contrast, do not change much, if at all. Yet other routes show a slow convergence towards some value. Even other routes slowly change around $y = 0.5$. Another effect that can be seen is that many routes share the same bottleneck link and therefore almost behave identically, which results in weight change lines that run almost in parallel.

In summary, we see that ReplEx converges quickly and without oscillations in realistic topologies under realistic workload traffic, while at the same time resulting in an improved network performance.

Table 1. IP throughput (TP) gains over the unoptimised network (unopt) for ReplEx, ReplEx without communication (ReplEx/noComm), traditional IGP weight optimisation (IGPWO), and combinations of these

Topology	Optimisation method	TP gain over unopt.
1755	IGPWO	10.7 %
1755	ReplEx	13.1 %
1755	ReplEx + IGPWO	11.2 %
1755	ReplEx/noComm	5.4 %
3967	IGPWO	6.5 %
3967	ReplEx	8.5 %
3967	ReplEx + IGPWO	7.6 %
3967	ReplEx/noComm	7.1 %
3967	ReplEx/noComm + IGPWO	7.0 %
6461	ReplEx	6.5 %
AT&T	ReplEx	16.4 %

6.3 Comparison to Traditional IGP Weight Optimisation

To demonstrate its potential, we compare ReplEx against a very popular TE method: the traditional technique of adjusting OSPF (or other IGP) weights [25]. The TOTEM traffic engineering toolbox [5] provides an implementation of this method, where it is called IGPWO ("IGP weight optimisation"). We adopt this terminology in the remainder of this document.

Note that IGPWO requires the full traffic matrix as input. The quality of its output is, of course, dependent on its input. In many cases, network operators only provide an estimation of the real traffic matrix, due to various technical limitations [29]. In our case however, we actually *do* know the traffic matrix with the *true* traffic demands, because we set up our simulation such that they are uniform; thus we can give the IGPWO algorithm a perfect input from which it can estimate the optimised OSPF weights.

We take the achieved IP throughput as an indicator for the network performance, as we have seen in our experiments that it is closely coupled with the packet loss rate and other performance measures [29,18]. Let us therefore have another look at Table 1. The first line in the first block shows that in the AS 1755 topology, the IGPWO method yields a 10.7 % increase in throughput over the completely unoptimised scenario (i. e., neither IGPWO weights, nor any ReplEx instances running). We can safely assume that this 10.7 % performance increase is not due to some random fluctuation, as the 95 % confidence interval[4] for the mean throughput with normal hop-count routing and the confidence interval for the mean throughput with IGPWO routing are clearly separated.

[4] This and the following confidence intervals are calculated from 32 group means of 125 measurements each. ACF plots showed no indication for autocorrelation-induced nonstationarity.

From the second line in the first block however, we can see that using ReplEx we can achieve an even higher gain of 13.1 % over the unoptimised scenario.

The first and second line of the second block in Table 1 compare the same values for the topology of AS 3967. Judging by the numbers to those for AS 1755, this topology obviously offers less optimisation potential, but nevertheless we get the same picture as before: traditional IGPWO yields a measurable performance improvement, but ReplEx yields an even better one.

6.4 Combining ReplEx and IGP Weight Optimisation

Recall that we can only apply ReplEx in networks where we can choose between multipath routes. A weight optimisation such as they are performed by IGPWO naturally reduces the number of available routes and therefore the possibilities among which ReplEx instances may choose to share their traffic. Nevertheless, the Totem IGPWO implementation does not necessarily remove all multipath routes from a scenario, but actually features a flag through which one can explicitly allow the use of multipath routes. We enabled this flag and then let run ReplEx on the network with the IGPWO OSPF weights. The result of this combination is shown in the third row of the first and second block of Table 1. In both of our simulated topologies, we note a small advantage for the combination of IGPWO and ReplEx over the pure IGPWO setting. Moreover, the performance gain is less than the one that can be achieved with ReplEx on a network with a simple hop-count routing.

6.5 Disabling Communication

For a number of reasons [29], it is interesting to see how ReplEx performs if we disable our distance-vector like information protocol altogether, i. e., let the ReplEx agents cast their decisions solely on their own local measurements. Thus a ReplEx instance cannot detect a bottleneck any longer if it is not an immediate outgoing link, but lies further downstream.

Recall that agents in the Wardrop model correspond not just to one single ReplEx instance at one router, but rather to *chains* of ReplEx instances along the agent-controlled paths from the source to the sink. If we disable communication between the ReplEx instances, we thus do not faithfully reproduce the actions of our Wardrop agents, and our theoretically proven properties (e. g., fast convergence, as described in Section 3.3) do not necessarily hold any more.

The fourth row of the first block in Table 1 reveals that we do indeed obtain a performance increase of 5.4 % over the unoptimised case; but this gain is significantly smaller than the ones that can be attained using IGPWO, or using standard ReplEx *with* inter-router signalling. For the 3967 topology, we obtain slightly different results: For this topology, ReplEx without communication leads to a performance improvement (Table 1, 2nd block, 4th row) that is comparable to the improvements achieved by IGPWO or by normal ReplEx involving communication. The throughput is even better than the throughput for IGPWO.

We conclude that even communicationless ReplEx yields a measurable increase in network throughput. This also holds when we combine it with traditional IGPWO traffic engineering (second block, fourth row). Closer inspection of the simulation results [29], however, reveals that these overall performance improvements are somewhat ambivalent, because they come at the price of reduced service quality in some parts of the network. Therefore, disabling communication seems to be a tradeoff whose advantage is questionable, whereas normal ReplEx with inter-router communication always proves to be advantageous.

7 Conclusions

We have seen that large populations of agents can "learn" a Wardrop equilibrium purely by simple selfish improvement rules. This even holds if the information that the agents rely upon may be imperfect, i. e., out of date. Furthermore, this convergence process happens quickly. We have shown that for a particular rerouting policy, the population converges towards a Wardrop equilibrium like an FPAS.

These results have put forth the design of two algorithms. First, a distributed algorithm to compute approximate Wardrop equilbria. The distributed computation time of this algorithm is independent of the network topology and the overall computation time is polynomial. Second, we have presented a traffic engineering protocol, ReplEx, which was simulated in various scenarios under realistic workloads and network topologies. The protocol converges quickly and significantly improves network throughput.

Further analyses and improvements to ReplEx are thinkable. Due to its genericity, ReplEx is not restricted to pure IP networks; thus the most interesting alley is to examine the performance gains when applying ReplEx to other types of networks. In general, our simulations involving realistic self-similar traffic with real TCP connections confirm that the game theoretic concept of Wardrop routing indeed can be applied in practical networking.

References

1. Ackermann, H., Röglin, H., Vöcking, B.: On the impact of combinatorial structure on congestion games. In: Proc. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 613–622 (2006)
2. Awerbuch, B., Azar, Y., Epstein, A.: The price of routing unsplittable flow. In: Proc. 37th Annual ACM Symposium on Theory of Computing, STOC (2005)
3. Awerbuch, B., Khandekar, R., Rao, S.: Distributed algorithms for multicommodity flow problems via approximate steepest descent framework. In: Proc. 18th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA (2007)
4. Awerbuch, B., Kleinberg, R.D.: Adaptive routing with end-to-end feedback: Distributed learning and geometric approaches. In: Proc. 36th Annual ACM Symposium on Theory of Computing (STOC), pp. 45–53 (2004)
5. Balon, S., Lepropre, J., Monfort, G.: TOTEM—TOolbox for Traffic Engineering Methods, `http://totem.run.montefiore.ulg.ac.be/`

6. Beckmann, M., McGuire, C.B., Winsten, C.B.: Studies in the Economics of Transportation. Yale University Press (1956)
7. Berenbrink, P., Friedetzky, T., Goldberg, L.A., Goldberg, P., Hu, Z., Martin, R.: Distributed selfish load balancing. In: Proc. 17th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA (2006)
8. Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computing: Numerical Methods. Athena Scientific (1989)
9. Blum, A., Even-Dar, E., Ligett, K.: Routing without regret: On convergence to Nash equilibria of regret-minimizing algorithms in routing games. In: Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC), Denver, Colorado, USA, July 2006, pp. 45–52. ACM, New York (2006)
10. Cao, J., Nyberg, C.: An approximate analysis of load balancing using stale state information for servers in parallel. In: Proc. 2nd IASTED International Conference on Communications Internet and Information Technology (November 2003)
11. Cao, Z., Wang, Z., Zegura, E.W.: Performance of hashing-based schemes for Internet load balancing. In: Proc. IEEE INFOCOM Conference (2000)
12. Christodoulou, G., Koutsoupias, E.: The price of anarchy of finite congestion games. In: Proc. 37th Annual ACM Symposium on Theory of Computing, STOC (2005)
13. Cole, R., Dodis, Y., Roughgarden, T.: How much can taxes help selfish routing? In: Proc. 4th ACM Conference on Electronic Commerce, pp. 98–107 (2003)
14. Cominetti, R., Correa, J.R., Moses, N.E.S.: Network games with atomic players. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 525–536. Springer, Heidelberg (2006)
15. Dreger, H., Feldmann, A., Mai, M., Paxson, V., Sommer, R.: Dynamic application-layer protocol analysis for network intrusion detection. In: Proc.15th Usenix Security Symposium (2006)
16. Even-Dar, E., Mansour, Y.: Fast convergence of selfish rerouting. In: Proc. 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), pp. 772–781 (2005)
17. Fabrikant, A., Papadimitriou, C., Talwar, K.: The complexity of pure Nash equilibria. In: Proc. 36th Annual ACM Symposium on Theory of Computing (STOC), pp. 604–612 (2004)
18. Fischer, S., Kammenhuber, N., Feldmann, A.: ReplEx—dynamic traffic engineering based on Wardrop routing policies. In: Proc. CoNext, Lisboa, Portugal (2006)
19. Fischer, S., Olbrich, L., Vöcking, B.: Approximating Wardrop equilibria with finitely many agents. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 238–252. Springer, Heidelberg (2007)
20. Fischer, S., Räcke, H., Vöcking, B.: Fast convergence to Wardrop equilibria by adaptive sampling methods. In: Proc. 38th Annual ACM Symposium on Theory of Computing (STOC), Seattle, WA, USA, May 2006, pp. 653–662. ACM, New York (2006)
21. Fischer, S., Vöcking, B.: On the evolution of selfish routing. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 323–334. Springer, Heidelberg (2004)
22. Fischer, S., Vöcking, B.: Adaptive routing with stale information. In: Proc. 24th Annual ACM Symposium on Principles of Distributed Computing (PODC), Las Vegas, NV, USA, pp. 276–283. ACM, New York (2005)
23. Fleischer, L.: Linear tolls suffice: New bounds and algorithms for tolls in single source networks. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 544–554. Springer, Heidelberg (2004)

24. Fortz, B., Rexford, J., Thorup, M.: Traffic engineering with traditional IP routing protocols. IEEE Communication Magazine, 118–124 (2002)
25. Fortz, B., Thorup, M.: Internet traffic engineering by optimizing OSPF weights. In: Proc. IEEE INFOCOM Conference (2000)
26. Fortz, B., Thorup, M.: Optimizing OSPF/IS-IS weights in a changing world. IEEE Journal on Selected Areas in Communications 20(4) (2002)
27. Haurie, A.B., Marcotte, P.: On the relationship between Nash-Cournot and Wardrop equilibria. Networks 15, 295–308 (1985)
28. Hayrapetyan, A., Tardos, É., Wexler, T.: The effect of collusion in congestion games. In: Proc. 38th Annual ACM Symposium on Theory of Computing (STOC), pp. 89–98 (2006)
29. Kammenhuber, N.: Traffic Adaptive Routing. Ph.D thesis, Technische Universität München (2008)
30. Khanna, A., Zinky, J.: The revised ARPANET routing metric. In: Proc. ACM SIGCOMM Conference (1989)
31. Khanna, A., Zinky, J.A.: The revised ARPANET routing metric. In: Proc. ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), September 1998, pp. 45–56. ACM, New York (1998)
32. Kurose, J.F., Ross, K.W.: Computer Networking, a top down approach featuring the Internet, 3rd edn. Addison-Wesley Longman, Amsterdam (2004)
33. Laor, M., Gendel, L.: The effect of packet reordering in a backbone link on application throughput. IEEE Network (September/October 2002)
34. Mitzenmacher, M.: How useful is old information? IEEE Transactions on Parallel and Distributed Systems 11(1), 6–20 (2000)
35. Park, K., Willinger, W. (eds.): Self-Similar Network Traffic and Performance Evaluation. Wiley-Interscience, Hoboken (2000)
36. Rexford, J.: Route optimization in IP networks. In: Handbook of Optimization in Telecommunications. Kluwer Academic Publishers, Dordrecht (2005)
37. Rosenthal, R.W.: A class of games possessing pure-strategy Nash equilibria. International Journal of Game Theory 2, 65–67 (1973)
38. Roughan, M., Thorup, M., Zhang, Y.: Traffic engineering with estimated traffic matrices. In: Proc. ACM Measurement Conference (2003)
39. Roughgarden, T.: Stackelberg scheduling strategies. In: Proc. 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 104–113 (2001)
40. Roughgarden, T.: The price of anarchy is independent of the network topology. Journal of Computer and System Sciences 67(2), 341–367 (2003)
41. Roughgarden, T.: The maximum latency of selfish routing. In: Proc. 15th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), pp. 973–974 (2005)
42. Roughgarden, T., Tardos, É.: How bad is selfish routing? Journal of the ACM 49(2), 236–259 (2002)
43. Savage, S., Collins, A., Hoffman, E., Snell, J., Anderson, T.E.: The end-to-end effects of internet path selection. In: Proc. ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), Boston, MA, pp. 289–299 (1999)
44. Spring, N., Mahajan, R., Wetherall, D.: Measuring ISP topologies with Rocketfuel. In: Proc. ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), Pittsburgh, PA, USA, August 2002, pp. 133–145. ACM, New York (2002)
45. Teixeira, R., Duffield, N., Rexford, J., Roughan, M.: Traffic matrix reloaded: Impact of routing changes. In: Proc. Passive and Active Measurement (2005)

46. Wang, H., Xie, H., Qiu, L., Yang, Y.R., Zhang, Y., Greenberg, A.: COPE: Traffic engineering in dynamic networks. In: Proc. ACM SIGCOMM Conference (2006)
47. Wardrop, J.G.: Some theoretical aspects of road traffic research. In: Proc. of the Institute of Civil Engineers, Pt. II, vol. 1, pp. 325–378 (1952)
48. Xiao, X., Hannan, A., Bailey, B., Ni, L.: Traffic engineering with MPLS in the Internet. IEEE Network Magazine (March 2000)
49. Zhang, C., Ge, Z., Kurose, J., Liu, Y., Towsley, D.: Optimal routing with multiple traffic matrices: Tradeoff between average case and worst case performance. In: Proc. 13th International Conference on Network Protocols, ICNP (2005)
50. Zhang, Y., Roughan, M., Lund, C., Donoho, D.: An information-theoretic approach to traffic matrix estimation. In: Proc. ACM SIGCOMM Conference (2003)

Models of Non-atomic Congestion Games – From Unicast to Multicast Routing

Lasse Kliemann and Anand Srivastav

Institut für Informatik
Christian-Albrechts-Universität Kiel
Christian-Albrechts-Platz 4
24118 Kiel
{lki,asr}@informatik.uni-kiel.de

Abstract. We give an overview of important results for non-atomic congestion games in their traditional form along with self-contained and short proofs and then present new results and challenges for an extended model, which we call *consumption-relevance congestion games*. We introduce the consumption-relevance congestion game first and show that all other models are special cases of it. Our focus is on the price of anarchy, computation of equilibria, and experimental studies. Further interesting aspects are summarized at the end of this article.

Keywords: non-atomic congestion games, Wardrop model, price of anarchy, unicast routing, multicast routing.

1 Introduction

1.1 Congestion Games

We give an informal introduction to congestion games and refer to Sec. 2 for exact definitions. Congestion games in general model competitive situations where *players* (or users) have to occupy one or more subsets from a set of *elements* (or resources) to accomplish their tasks. These subsets of elements are called *strategies*. The tasks of each user could be to transmit one or more messages through a communication network, in which case the elements could be the edges of the network and the strategies could constitute paths, trees, or collections of paths. The *latency* (or cost, disutility) experienced by each user depends on the *congestion* of the occupied elements, on the elements themselves, and possibly also on the strategy through which the occupation takes place. The congestion of an element in turn depends on the total amount of users that occupy that element, and again possibly also on the strategies through which the occupations take place – but not on any form of 'identity' of the users.

Congestion games have been used to model classical point-to-point routing, also known as unicast routing. There are limitations to the traditional model. Traditionally, there are only limited ways to express the role of an element inside a particular strategy. In one of the previously most general models, studied by Roughgarden and Tardos [1], elements may have different sensitivity to

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 292–318, 2009.

occupation through different strategies. This is expressed by element-strategy-dependent factors called *rates of consumption.* The same factor that influences sensitivity to occupation of an element e via a strategy S also influences how much the latency of e will count for the players that choose strategy S. Hence, players choosing a sensitive element are charged doubly for this. *With these limitations, traditional congestion games cannot model many complex routing applications, e.g., with bottleneck links or multicast routing.* In order to help the theory of congestion games to catch up with the requirement to model more complex routing, we introduce an extended model, which we call *consumption-relevance congestion games.*

1.2 An Introductory Example

We informally present a simple example, in order to give the reader a more concrete idea of the matter. The example is essentially due to Pigou[1] [2]. It does not require to leave the traditional model; it is however suited to illustrate the basic idea of congestion games and the price of anarchy.

Consider two cities s and t, connected by two highways H_1 and H_2. Many commuters have to travel from s to t at the same time, and each of them can choose between H_1 and H_2; these constitute the strategies, which in this simple example consist of only one element each, namely the respective highway. Highway H_1 is rather long, but broad enough so that no congestion effects occur: no matter how many commuters choose highway H_1, their travel time – the latency – will always be one hour. Highway H_2 is much shorter, but also narrower. Commute time on H_2 is influenced by congestion effects: the time grows proportional to the number of drivers choosing H_2, and it equals one hour if all drivers choose H_2 (and none chooses H_1). Let us normalize the total amount of drivers to 1, and say that the commute time experienced by drivers on H_2 takes x times one hour if x drivers take that highway, $x \in [0, 1]$.

If $x < 1$, then the drivers on H_2 experience less commute time than those on H_1, and so we expect that as long as $x < 1$, the $1 - x$ drivers on H_1 see an incentive in switching to H_2. It follows that the system is stable – we also say: in equilibrium – if and only if $x = 1$. Intriguingly, *no-one* is then in a better position than before. To quantify the overall performance of the system, we use the average travel time, which is $1 - x + x^2$. It is minimum, namely $\frac{3}{4}$, when $x = \frac{1}{2}$; it is 1 when $x = 1$, i.e., in equilibrium. The ratio between the two, which is $\frac{4}{3}$ here, is called the price of anarchy. It is called that way, for, if

[1] The description given by Pigou [2, p. 194] reads:

> "Suppose there are two roads, ABD and ACD both leading from A to D. If left to itself, traffic would be so distributed that the trouble involved in driving a "representative" cart along each of the two roads would be equal. But, in some circumstances, it would be possible, by shifting a few carts from route B to route C, greatly to lessen the trouble of driving those still left on B, while only slightly increasing the trouble of driving along C."

a central authority for minimizing the average travel time was in charge, that authority would enforce $x = \frac{1}{2}$, i.e., one half of the drivers would be allowed to take highway H_2, and the other half would be *forced* to take highway H_1 and accept that those on highway H_2 are better off.

1.3 Organization

The rest of this survey is organized as follows. In the next section we define the non-atomic consumption-relevance congestion game model (NCRCG) and explain the game-theoretic background. In Sec. 3 we show how unicast routing can be modeled as an NCRCG and illustrate that by a simple example. Two matrices, the matrix of consumption numbers C and the matrix of relevance numbers R, play an important role in the NCRCG model, where the case $C = R$ coincides with the known model of non-atomic congestion games (NCG) [1]. It will become clear in Sec. 3 that unicast can be modeled with $C = R$, i.e., as a NCG. In Sec. 4 we treat further aspects of NCGs: known results on the uniqueness of Nash equilibria and bounds on the price of anarchy are presented with concise proofs. We also briefly sketch computational issues. Sec. 5 presents applications and new results for NCRCGs; $C \neq R$ will be allowed in that section. The connection to multicast routing is explained and illustrated by an example. Lower and upper bounds on the price of anarchy are stated and new computational challenges are discussed. Numerical results for random instances are given. Finally, in Sec. 6 we summarize further interesting topics not covered by this survey.

2 The Model of Non-atomic Consumption-Relevance Congestion Games

We will introduce the general model at the beginning. This will allow a more fluent presentation and in particular show how all other models constitute special cases of the general one.

2.1 Basic Notation: Elements, Strategies, and Latencies

Consider the following items:

(i) A number $m \in \mathbb{N}$. We denote $E := [m]$ $(= \{1, \dots, m\})$ and say that each number in E stands for an *element* (or a *resource*).

(ii) For each element $e \in E$ a function $\ell_e : \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0}$. These functions are called the *element latency functions.* We require them to be continuous and non-decreasing.

(iii) A number $N \in \mathbb{N}$. Each number in $[N]$ stands for a *player class.*
The model is feasibly defined (and interesting) even with one player class, i.e., $N = 1$. In fact, all examples given in this article only have one player class.

(iv) For each $i \in [N]$ a number $d_i \in \mathbb{R}_{>0}$, called the *demand* of player class i.

(v) For each $i \in [N]$ a finite set $\mathfrak{S}_i$, chosen such that $\mathfrak{S}_1, \dots, \mathfrak{S}_N$ are pairwise disjoint and each $\mathfrak{S}_i$ has at least cardinality 2. We say that each element in $\mathfrak{S}_i$ stands for a *strategy* for player class i. Denote $\mathfrak{S} := \bigcup_{i \in [N]}$ and $n := |\mathfrak{S}|$. We will often identify $\mathfrak{S} = [n]$ to simplify notation.

(vi) A matrix $C \in \mathbb{R}_{\geq 0}^{m \times n}$ – its entries are called *consumption numbers* – and a matrix $R \in \mathbb{R}_{\geq 0}^{m \times n}$ – its entries are called *relevance numbers*. We say that for $e \in E$ and $S \in \mathfrak{S}$ the number $C_{e,S}$ is the consumption of element e under strategy S, and the number $R_{e,S}$ is the relevance of element e for strategy S. We further demand that neither of the two matrices has row or a column of only zeros, and that

$$C_{e,S} = 0 \text{ if and only if } R_{e,S} = 0$$

for all $e \in E$, $S \in \mathfrak{S}$.

These items (i) to (vi) define an instance of the NCRCG model.

The set of *action distributions* is

$$\mathcal{A} := \{a \in \mathbb{R}_{\geq 0}^{n};\ \sum_{S \in \mathfrak{S}_i} a_S = d_i \quad \forall i \in [N]\}.$$

An action distribution describes a way of how to distribute the demand d_i of each player class i across the strategies in $\mathfrak{S}_i$.

Let $S \in \mathfrak{S}$. The set of those elements that have non-zero entries in column S of matrix C (or, equivalently, R), will – abusing notation – often also be denoted by S, and we also speak of an element being 'contained' in S. Keep in mind that two strategies with different consumption or relevance numbers can constitute the same set of elements in this sense. This will be no problem; we will always treat such strategies as distinct. Moreover, it is sometimes convenient to consider the set of all strategies which 'contain' a particular element e. We will denote that as $\mathfrak{S}(e) := \{S \in \mathfrak{S};\ C_{e,S} \neq 0\}$, which is the same as $\{S \in \mathfrak{S};\ R_{e,S} \neq 0\}$.

Given an action distribution a, we define the *vector of congestions* as $\boldsymbol{g}(a) := C \cdot a$, where the dot denotes the usual matrix-vector product. Hence we have $g_e(a) = \sum_{S \in \mathfrak{S}} C_{e,S} a_S = \sum_{S \in \mathfrak{S}(e)} C_{e,S} a_S$. The congestion is a measure of how heavily an element is stressed. As can be noticed from that definition, the model allows that an element may be stressed differently by the same amount of users *depending on the strategy through which these users occupy the element.*

Recall that each element has a latency function $\ell_e : \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0}$. This functions tells us how the element reacts to congestion. The quantity $\ell_e(g_e(a))$ is called the *element latency* of e under a.

Finally we have to define how that latency is experienced by the users. Since we study the non-atomic case, we think of the set of players as a continuum,[2] represented for each player class i by the interval $[0, d_i]$. Each action distribution describes a possible way of how this interval can be partitioned and the parts assigned to strategies. All the players which by this are assigned to a strategy

[2] The game-theoretic background of this will be elaborated on in the next section.

S experience the latency $L_S(a) := \sum_{e \in E} R_{e,S} \ell_e(g_e(a))$, which, thinking of S as a subset of E, can be written as $\sum_{e \in S} R_{e,S} \ell_e(g_e(a))$. We call this the *strategy latency* of S under a. Since R is a $(m \times n)$-matrix and $\boldsymbol{\ell}(\boldsymbol{g}(a)) := (\ell_e(g_e(a)))_{e \in E}$ a vector[3] of length m, we can write the vector of all strategy latencies in compact form using matrix-vector multiplication: $\boldsymbol{L}(a) = (\boldsymbol{\ell}(\boldsymbol{g}(a))^{\perp} \cdot R)^{\perp} \in \mathbb{R}^n_{\geq 0}$.

2.2 Social Cost and Nash Equilibria

The *social cost* of an action distribution a is the sum of all strategy latencies weighted with the amount of players experiencing the particular latency:

$$\mathsf{SC}(a) := \sum_{S \in \mathfrak{S}} L_S(a) a_S.$$

By an optimal action distribution we will refer to one that has minimum social cost. The minimum social cost is denoted by OPT; we assume that $\mathsf{OPT} > 0$. Optimal action distributions always exist, since SC is continuous (recall that each ℓ_e is continuous) and $\mathcal{A}$ is a compact set. Related to the social cost and important for later proofs is the *total element cost*

$$\mathsf{EC}(a) := \sum_{e \in E} \ell_e(g_e(a)) g_e(a).$$

In general, it differs from the social cost. However, in the special case of $C = R$, a simple calculation shows that we have $\mathsf{SC}(a) = \mathsf{EC}(a)$ for all $a \in \mathcal{A}$.

The *mixed social cost* and *mixed total element cost* for two action distributions $a, \tilde{a}$ will also play an important role, and we define them here for later:

$$\mathsf{SC}^a(\tilde{a}) := \sum_{S \in \mathfrak{S}} L_S(a) \tilde{a}_S \quad \text{and} \quad \mathsf{EC}^a(\tilde{a}) := \sum_{e \in E} \ell_e(g_e(a)) g_e(\tilde{a}).$$

Again, in the special case of $C = R$, we have $\mathsf{SC}^a(\tilde{a}) = \mathsf{EC}^a(\tilde{a})$ for all $a, \tilde{a} \in \mathcal{A}$.

When all players are assigned to minimum-latency strategies by an action distribution a, we say that a is a *Nash equilibrium.* More formally, a is a Nash equilibrium if

$$\big(\, a_S > 0 \Longrightarrow L_S(a) \leq L_T(a) \,\big) \quad \forall S, T \in \mathfrak{S}_i \quad \forall i \in [N]. \tag{1}$$

Directly from that definition it follows that for a Nash equilibrium a we have

$$\mathsf{SC}(a) = \sum_{i \in [N]} \Lambda_i(a) d_i, \tag{2}$$

where $\Lambda_i(a) := \min\{L_S(a);\ S \in \mathfrak{S}_i\}$ for each $i \in [N]$.

[3] We use boldface for vectors of congestions, element latencies, and strategy latencies. So, we write $\boldsymbol{\ell}(\cdot)$ for the vector of all element latency functions, whereas we sometimes use $\ell(\cdot)$ to denote a single element latency function.

It is sometimes misunderstood that (1) was to model a competition of the different player classes against each other. The truth is that in the model at hand, the player classes exist only to allow players with different demands and different sets of eligible strategies in the game, and that the model is feasibly defined even with only one player class.

Informally speaking, Nash equilibria characterize those action distributions where there is no way for a 'single player' to unilaterally improve his latency. We think of each 'single player' to constitute a so small fraction of the total demand that his decision *does not change the system (i.e., congestions and latencies), but only changes the latency he experiences when using the chosen strategy, which is already given.* This is the *non-atomic model*, also known as the *Wardrop model* [3]. An illustrative example for this is road traffic. The decision of a single driver which route to take will have *virtually no effect* on the congestions and latencies of the road network. However, each driver will only choose such routes that promise minimum latency. Hence, this system can be considered stable once that all drivers have chosen minimum-latency routes. Another example is a communication network with a large number of users where each user has only got a small amount of data to send or receive. For further historic background, we quote Wardrop [3, p. 344–345]:

> The problem is to discover how traffic may be expected to distribute itself over alternative routes, and whether the distribution adopted is the most efficient one. [...] Consider two alternative criteria [...] which can be used to determine the distribution on the routes, as follows:
>
> (1) The journey times on all the routes actually used are equal, and less than those that would be experienced by a single vehicle on any unused route.
>
> (2) The average journey time is a minimum.
>
> The first criterion is quite a likely one in practice, since it might be assumed that traffic will tend to settle down into an equilibrium situation in which no driver can reduce his journey time by choosing a new route. On the other hand, the second criterion is the most efficient in the sense that it minimizes the vehicle-hours spent on the journey.

Our definitions of Nash equilibria and optimal action distributions, respectively, match these criteria.

Existence and Variational Inequality Formulation

Theorem 1. *An action distribution a is a Nash equilibrium if and only if*

$$\mathsf{SC}(a) \leq \mathsf{SC}^{a}(\tilde{a}) \quad \forall \tilde{a} \in \mathcal{A}. \tag{VAR}$$

Note that $\mathsf{SC}(a) = \mathsf{SC}^{a}(a)$. Condition (VAR) is a well-studied variational inequality. The result of this theorem has already been noted by Smith [4] and used in many different contexts. We give a proof for completeness. The proof does not use the notion of elements or congestions (nor do (1) or (VAR)) and hence this result is valid in an even broader context (not discussed in this work).

Proof (Thm. 1). Let first a be a Nash equilibrium. Fix an arbitrary $\tilde{a} \in \mathcal{A}$. By (2) we have

$$\begin{aligned}\mathsf{SC}^a(\tilde{a}) - \mathsf{SC}(a) &= \sum_{i\in[N]} \sum_{S\in\mathfrak{S}_i} \underbrace{L_S(a)}_{\geq \Lambda_i(a)} \tilde{a}_S - \sum_{i\in[N]} \Lambda_i(a) d_i \\ &\geq \sum_{i\in[N]} (\Lambda_i(a) \underbrace{\sum_{S\in\mathfrak{S}_i} \tilde{a}_S}_{=d_i} - \Lambda_i(a) d_i) = 0.\end{aligned}$$

Now, let $a \in \mathcal{A}$ such that (VAR) holds. Fix $i_0 \in [N]$ and pick $S_0 \in \mathfrak{S}_{i_0}$ such that $L_{S_0}(a) = \Lambda_{i_0}(a) =: \Lambda$. Define $\tilde{a} \in \mathbb{R}^n_{\geq 0}$ by

$$\tilde{a}_S := \begin{cases} d_{i_0} & \text{if } S = S_0 \\ 0 & \text{if } S \in \mathfrak{S}_{i_0} \setminus \{S_0\} \\ a_S & \text{otherwise} \end{cases} .$$

Then $\tilde{a}$ is an action distribution, and by (VAR) we have

$$\begin{aligned} 0 &\leq \sum_{i\in[N]} \sum_{s\in\mathfrak{S}_i} L_S(a) \underbrace{(\tilde{a}_S - a_S)}_{=0 \text{ if } i \neq i_0} = \sum_{S\in\mathfrak{S}_{i_0}} L_S(a)(\tilde{a}_S - a_S) \\ &= \underbrace{L_{S_0}(a)}_{=\Lambda} d_{i_0} - \sum_{S\in\mathfrak{S}_{i_0}} \underbrace{L_S(a)}_{\geq \Lambda} a_S. \end{aligned}$$

Since $\sum_{S\in\mathfrak{S}_{i_0}} a_S = d_{i_0}$, it follows that $L_S(a) = \Lambda$ whenever $a_S > 0$, $S \in \mathfrak{S}_{i_0}$. Because i_0 was taken arbitrarily, a is a Nash equilibrium. □

By this variational inequality formulation, we get an existence result.

Theorem 2. *Every NCRCG admits at least one Nash equilibrium.*

Proof. Solutions to (VAR) exist since $L_S(\cdot)$ is continuous for each $S \in \mathfrak{S}$ and $\mathcal{A}$ is compact and convex; see, e.g., [5, Thm. 1.4] or [6, Thm. 3.1]. □

We denote the set of all Nash equilibria for a given NCRCG by $\mathcal{N} \subseteq \mathcal{A}$.

Condition (VAR) has an interesting interpretation, supplementing the previous discussion. The mixed social cost $\mathsf{SC}^a(\tilde{a})$ gives us a kind of 'social cost' dependent on two parameters: the first one, namely a, defines the *system* (i.e., congestions and corresponding element and strategy latencies), and the second one, namely $\tilde{a}$, indicates how that system is used. Condition (VAR) says that Nash equilibria are exactly those action distributions a that give the optimal way to use the system that is fixed by a.

Prices of Anarchy and Stability. The relation of Nash equilibria to optimal action distributions is captured by the following two quantities.

- The *price of anarchy* (or *coordination ratio*) $\sup_{a\in\mathcal{N}} \frac{\mathsf{SC}(a)}{\mathsf{OPT}}$.

- The *price of stability* $\inf_{a\in\mathcal{N}} \frac{\mathsf{SC}(a)}{\mathsf{OPT}}$.

In the next section we will explain and illustrate the introduced concepts by an example. The reader might also like to take an excursion to Sec. 5.1 for another example.

3 Unicast Routing

Let $G = (V, E)$ be a directed multigraph modeling a communication network with continuous, non-decreasing latency functions $\ell_e : \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0}$ for each edge e and N source-sink pairs (s_i, t_i), $i = 1, \ldots, N$. Each of these N pairs has got a total demand of d_i units of flow to route from s_i to t_i, which represents a continuum $[0, d_i]$ of players. The latency for transmitting one unit of flow along any path in the graph is the sum of the latencies of the edges in the path.

We can model this as an NCRCG in a straight-forward way. The edges of the graph together with their latency functions become the elements and element latency functions of the NCRCG, respectively. The N source-sink pairs correspond to N player classes. The set of strategies for each player class is a set of s_i-t_i-paths and hence we can take the corresponding edge-path incidence matrix, which is a 0/1-valued matrix, as the consumption matrix C as well as the relevance matrix R. A Nash equilibrium now means that all flow travels along minimum-latency paths.

Some interesting results for unicast routing will not be mentioned in this section. This is because they also hold in the more general context of $C = R$, and hence we defer them to the Sec. 4.

3.1 Generalized Version of Pigou's Example

Recall Pigou's example from Sec. 1.2. We consider a generalization with a parameter $p \in \mathbb{N}_{\geq 1}$, as depicted in Fig. 1, and describe it in terms of the NCRCG model. We have one player class with source s and sink t, which are connected by two different edges. One of them has latency function constant 1 and the other has latency function $x \mapsto x^p$, where $p \in \mathbb{N}_{\geq 1}$ is some fixed number. Let the demand be $d = 1$. The two available strategies are obvious: players can choose to take the one edge or the other to reach t, so one strategy consists of edge 1 (with latency function ℓ_1) and the other of edge 2 (with latency function ℓ_2).

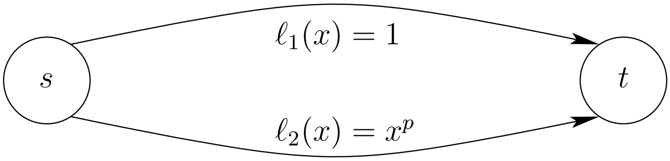


Fig. 1. Pigou's example with parameter p

Up to different enumerations of strategies and elements, the matrices C and R look like this:

$$C = R = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Let us compute the strategy latencies and social cost for some arbitrary action distribution $a = (a_1, a_2)$.

$$\begin{aligned} L_1(a) &= R_{1,1}\ell_1(g_1(a)) + R_{2,1}\ell_2(g_2(a)) = 1 \cdot \underbrace{\ell_1(g_1(a))}_{=1} + 0 \cdot \ell_2(g_2(a)) = 1, \\ L_2(a) &= R_{1,2}\ell_1(g_1(a)) + R_{2,2}\ell_2(g_2(a)) = 0 \cdot \ell_1(g_1(a)) + 1 \cdot \ell_2(g_2(a)) \\ &= \ell_2(C_{2,1}a_1 + C_{2,2}a_2) = \ell_2(a_2) = a_2^p, \\ \mathsf{SC}(a) &= L_1(a)a_1 + L_2(a)a_2 = a_1 + a_2^{p+1} \overset{a_1+a_2=1}{=} 1 - a_2 + a_2^{p+1}. \end{aligned}$$

We see immediately that the only Nash equilibrium is $a = (0, 1)$, for if any fraction of the flow was to take the first strategy, say, $a' = (\varepsilon, 1 - \varepsilon)$, we would have $L_2(a') < 1 = L_1(a')$, meaning that not all flow would travel along minimum-latency strategies. On the other hand, simple real calculus gives that $a^* := (1 - (p + 1)^{-1/p}, (p + 1)^{-1/p})$ is the only optimal action distribution. Its social cost is

$$\begin{aligned} \mathsf{SC}(a^*) &= 1 - (p + 1)^{-1/p} + (p + 1)^{-(p+1)/p} \\ &= 1 - ((p + 1)(p + 1)^{-\frac{1}{p}-1} - (p + 1)^{-(p+1)/p}) \\ &= 1 - (p(p + 1)^{-\frac{1}{p}-1} + \underbrace{(p + 1)^{-\frac{1}{p}-1} - (p + 1)^{-(p+1)/p}}_{=0}) \qquad (3) \\ &= 1 - p(p + 1)^{-(p+1)/p}. \end{aligned}$$

This tends to 0 as p tends to ∞. Hence, the price of anarchy – as well as the price of stability – is $\frac{\mathsf{SC}(a)}{\mathsf{SC}(a^*)} = (1 - p(p + 1)^{-(p+1)/p})^{-1} = \Theta(\frac{p}{\ln p})$, which tends to ∞ as p tends to ∞.

3.2 Anarchy Value

The Pigou example allows two observations: first, we see that arbitrary high prices of anarchy can be achieved with very simple networks – just one source, one sink, and parallel edges in between. Second, we observe a correlation between the price of anarchy and the kind of latency functions used – the higher the degree of the polynomial ℓ_2, the larger the price of anarchy becomes.

In fact, Roughgarden [7] shows that the price of anarchy in unicast routing can be upper-bounded by a value only dependent on the class $\mathcal{L}$ of eligible edge latency functions, the so-called *anarchy value* $\alpha(\mathcal{L})$. The only additional assumptions that he makes are: each $\ell \in \mathcal{L}$ is differentiable, $x \mapsto \ell(x)x$ is a convex mapping, and $\mathcal{L}$ contains at least one non-zero function. Such a class $\mathcal{L}$ is called *standard*.[4]

[4] See [7, Def. 2.3.5 and 3.3.1].

He furthermore shows that the $\alpha(\mathcal{L})$-bound on the price of anarchy is tight from a worst-case point of view. More precisely: let $\varepsilon > 0$ and let $\mathcal{L}$ be a standard class that is also *diverse*, i.e., for each $y \in \mathbb{R}_{\geq 0}$ contains a function ℓ such that $\ell(0) = y$. Then there exists a unicast network consisting of one source, one sink and parallel edges in between with latency functions drawn from $\mathcal{L}$ such that its price of anarchy is at least $\alpha(\mathcal{L}) - \varepsilon$. Hence, *from a worst-case point of view*, the price of anarchy in unicast routing is independent of the network topology. On the other hand, for a particular instance, it may be dependent on the network topology; adding or removing edges may change the price of anarchy. A prominent example for this is Braess's Paradox [8,9] (see also [7, Sec. 1.2.2]), where an additional edge worsens the price of anarchy.

We will revisit the anarchy value in more detail in Sec. 4.1, and a corresponding simplification (and generalization at the same time) in Sec. 4.2.

4 The Case $C = R$

In the previous section, only entries 0 and 1 were allowed in the matrices C and R, and we had $C = R$. Now we allow arbitrary values, but still require that $C = R$. This model is usually referred to as the model of *non-atomic congestion games* (NCG for short) in the literature [1].

As mentioned earlier, the case $C = R$ has the nice property that mixed social cost and mixed total element cost coincide, i.e., $\mathsf{EC}^a(\tilde{a}) = \mathsf{SC}^a(\tilde{a})$ for all action distributions $a, \tilde{a}$. Moreover, all Nash equilibria have the same social cost, and hence price of anarchy and price of stability coincide. We give a short proof that all Nash equilibria have the same social cost.

Theorem 3. *Let $a, a' \in \mathcal{N}$. Then*

(i) $\ell_e(g_e(a)) = \ell_e(g_e(a'))$ (and hence $g_e(a) = g_e(a')$ if ℓ_e is strictly increasing),
(ii) $\mathsf{SC}(a) = \mathsf{SC}(a')$.

Proof. Using (VAR), we get $\mathsf{SC}(a) - \mathsf{SC}^a(a') \leq 0$ and $\mathsf{SC}(a') - \mathsf{SC}^{a'}(a) \leq 0$, so

$$\begin{aligned}
0 &\geq \mathsf{SC}(a) - \mathsf{SC}^a(a') + \mathsf{SC}(a') - \mathsf{SC}^{a'}(a) \\
&= \sum_{S \in \mathfrak{S}} (L_S(a) - L_S(a'))(a_S - a'_S) \\
&= \sum_{S \in \mathfrak{S}} \sum_{e \in S} R_{e,S}(\ell_e(g_e(a)) - \ell_e(g_e(a')))(a_S - a'_S) \\
&= \sum_{e \in E} (\ell_e(g_e(a)) - \ell_e(g_e(a'))) \sum_{S \in \mathfrak{S}(e)} R_{e,S}(a_S - a'_S) \\
&= \sum_{e \in E} (\ell_e(g_e(a)) - \ell_e(g_e(a'))) \sum_{S \in \mathfrak{S}(e)} C_{e,S}(a_S - a'_S) \qquad \text{since } C = R \\
&= \sum_{e \in E} (\ell_e(g_e(a)) - \ell_e(g_e(a')))(g_e(a) - g_e(a')).
\end{aligned}$$

The last equation was positive if for some e we had $\ell_e(g_e(a)) \neq \ell_e(g_e(a'))$; recall that element latency functions are non-decreasing. This shows (i). It follows furthermore that $L_S(a) = L_S(a')$ for all $S \in \mathfrak{S}$ and also that $\Lambda_i(a) = \Lambda_i(a')$ for all $i \in [N]$. Assertion (ii) now follows with (2). □

Note how in the proof the fact that $C = R$ allows us to draw a bow from strategy latencies to congestions.

For the rest of this section, we will elaborate on two topics, namely upper-bounding the price of anarchy by the anarchy value and computation of Nash equilibria. The next section, 4.1, may be challenging to grasp on a first read, and the reader is encouraged to skim through this section first and continue in detail in Sec. 4.2. These two sections, 4.1 and 4.2, present a similar matter. They are arranged in historic order.

4.1 Upper-Bounding the Price of Anarchy with the Anarchy Value

Let $\mathcal{L}$ be a standard class of element latency functions. For each $\ell \in \mathcal{L}$ define the *marginal cost function* $\ell^* : \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0}, x \mapsto \ell'(x)x + \ell(x)$. Choose a function $\mu_\ell : \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0}$ such that

$$\ell^*(\mu_\ell(x)) = \ell(x) \quad \text{and} \quad \mu_\ell(x) \leq x \quad \text{for all} \quad x \in \mathbb{R}_{\geq 0}. \tag{4}$$

Such a function exists as we will see in Sec. 4.2, but needs not to be unique. We will also write μ_e instead of μ_{ℓ_e}. We are now ready to define the *anarchy value* [1, Def. 4.3] of $\mathcal{L}$, denoted $\alpha(\mathcal{L})$. Using the convention $0/0 := 0$, we define

$$\alpha(\mathcal{L}) := \sup_{\substack{\ell \in \mathcal{L} \\ v \geq 0}} \frac{\ell(v)v}{(\ell(\mu_\ell(v)) - \ell(v))\mu_\ell(v) + \ell(v)v}. \tag{5}$$

Since $\mathcal{L}$ is standard, which means in particular that it contains a non-zero function, we always have $\alpha(\mathcal{L}) > 0$. One can also prove that this definition is independent of the actual choices of the μ_ℓ functions. This also will become clear in Sec. 4.2 by essentially the same simple argument which we will use for the existence of μ_ℓ.

Although the expression defining the anarchy value may look complicated at first, anarchy values for many interesting classes of functions can be derived relatively easily, see [7, Sec. 3.5.2]. For instance, let $\text{Poly}^+(p)$ be the class of polynomials of degree at most p and non-negative coefficients. Then

$$\alpha(\text{Poly}^+(p)) = (1 - p(p+1)^{-(p+1)/p})^{-1} = \Theta(\frac{p}{\ln p}), \tag{6}$$

exposing Pigou's example as a worst-case scenario. This also shows that the anarchy value may be ∞, e.g., if we consider the set of all polynomials with non-negative coefficients.

Correa, Schulz, and Stier Moses [10] show that the concept of anarchy value can be simplified and generalized at the same time. Yet, we will first show how to work with the anarchy value by presenting a compact proof for the upper bound along the lines of Roughgarden and Tardos. In the next section, we will present the extensions of [10] together with a formal derivation of (6).

Theorem 4 (Roughgarden, Tardos [1]). *Let $\mathcal{L}$ be a standard class of element latency functions. The price of anarchy of any NCG with element latency functions drawn from $\mathcal{L}$ is no more than $\alpha(\mathcal{L})$.*

Proof. For any three reals $x, y, z \in \mathbb{R}_{\geq 0}$ and elements $e \in E$ we have

$$\begin{aligned}
&\ell_e(y)y = \ell_e(z)z + \int_z^y \ell_e^*(t)dt \geq \ell_e(z)z + \ell_e^*(z)(y-z) \\
&= (\ell_e(z) - \ell_e^*(z))z + \ell_e(x)x + \ell_e^*(z)y - \ell_e(x)x \\
&= (\ell_e(\mu_e(x)) - \ell_e^*(\mu_e(x)))\mu_e(x) + \ell_e(x)x + \ell_e^*(\mu_e(x))y - \ell_e(x)x \quad \text{put } z := \mu_e(x) \\
&= (\ell_e(\mu_e(x)) - \ell_e(x))\mu_e(x) + \ell_e(x)x + \ell_e(x)y - \ell_e(x)x \qquad \text{definition of } \mu_e \\
&\geq \frac{1}{\alpha(\mathcal{L})}\ell_e(x)x + \ell_e(x)y - \ell_e(x)x.
\end{aligned}$$

Let a be a Nash equilibrium and a^* an optimal action distribution. Evaluating the above for $x := g_e(a)$ and $y := g_e(a^*)$ and taking the sum over all $e \in E$ yields

$$\begin{aligned}
\mathsf{EC}(a^*) &= \sum_{e \in E} \ell_e(g_e(a^*))g_e(a^*) \\
&\geq \frac{1}{\alpha(\mathcal{L})} \sum_{e \in E} \ell_e(g_e(a))g_e(a) + \sum_{e \in E} \ell_e(g_e(a))g_e(a^*) - \sum_{e \in E} \ell_e(g_e(a))g_e(a) \\
&= \frac{1}{\alpha(\mathcal{L})}\mathsf{EC}(a) + \mathsf{EC}^a(a^*) - \mathsf{EC}(a).
\end{aligned}$$

Since we are in the case $C = R$, we have that (mixed) social cost equals (mixed) total element cost, and so

$$\mathsf{SC}(a^*) \geq \frac{1}{\alpha(\mathcal{L})}\mathsf{SC}(a) + \mathsf{SC}^a(a^*) - \mathsf{SC}(a) \overset{\text{by (VAR)}}{\geq} \frac{1}{\alpha(\mathcal{L})}\mathsf{SC}(a).$$

This finishes the proof, since we now have $\frac{\mathsf{SC}(a)}{\mathsf{SC}(a^*)} \leq \alpha(\mathcal{L})$. □

4.2 The Parameter $\beta(\mathcal{L})$

Following [10], we present an extension of the concept of anarchy value, which acts as a simplification at the same time. It is also applicable for tightly bounding the price of anarchy in unicast routing where each edge in addition to its latency functions has got a *capacity*, which is the main theme of [10], but will not be treated here.

Let $\mathcal{L}$ be a class of continuous, non-decreasing functions $\mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0}$, not necessarily standard. Using the convention $0/0 := 0$, define

$$\beta(\mathcal{L}) := \sup_{\substack{\ell \in \mathcal{L} \\ v \geq 0}} \frac{1}{\ell(v)v} \max_{x \geq 0}\{(\ell(v) - \ell(x))x\}. \tag{7}$$

Monotonicity ensures that there is no division by zero, except 0/0, which is defined to 0. It follows that

$$(\ell(v) - \ell(x))x \leq \beta(\mathcal{L})\, \ell(v)v \quad \forall v, x \in \mathbb{R}_{\geq 0} \quad \forall \ell \in \mathcal{L}. \tag{8}$$

It is easy to see that always $\beta(\mathcal{L}) \leq 1$. In case that $\mathcal{L}$ happens to be a standard class with finite anarchy value $\alpha(\mathcal{L})$, we have $\beta(\mathcal{L}) < 1$ and

$$\frac{1}{1 - \beta(\mathcal{L})} = \alpha(\mathcal{L}). \tag{9}$$

This is seen as follows: assume that $\mathcal{L}$ is standard and fix $\ell \in \mathcal{L}$, $v \geq 0$, and define the function $\phi : \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}, x \mapsto (\ell(v) - \ell(x))x$, which occurs in (7). Since $\phi(0) = \phi(v) = 0$, and, by monotonicity of ℓ, $\phi(x) \geq 0$ for all $x \leq v$, and $\phi(x) \leq 0$ for all $x \geq v$, ϕ attains its global maximum in some $x^* \in (0, v)$. Real calculus gives that $0 = \phi'(x^*) = \ell(v) - \ell'(x^*)x^* - \ell(x^*) = \ell(v) - \ell^*(x^*)$, hence $\ell^*(x^*) = \ell(v)$. As a first result, this proves the existence of the function μ_ℓ, cf. (4). Conversely, since $\mathcal{L}$ is standard and so ϕ is concave, any point x^* with $\ell^*(x^*) = \ell(v)$ maximizes ϕ.

Now, turning to the definition of anarchy value, displayed in (5), we reformulate and get $1 - \frac{1}{\alpha(\mathcal{L})} = \sup_{\ell \in \mathcal{L}, v \geq 0} \frac{1}{\ell(v)v}(\ell(v) - \ell(\mu_\ell(v)))\mu_\ell(v)$. A function of the same form as ϕ above occurs in that expression. From what we have shown, we know that for fixed ℓ and v we have $(\ell(v) - \ell(\mu_\ell(v)))\mu_\ell(v) = \max_{x \geq 0}(\ell(v) - \ell(x))x$, since $\ell^*(\mu_\ell(v)) = \ell(v)$. This for one proves (9) and also shows that (5) is independent of the actual choices for μ_ℓ, $\ell \in \mathcal{L}$.

A short proof for a bound on the price of anarchy including the result from Thm. 4 is possible using the parameter β.

Theorem 5 (Correa, Schulz, Stier Moses [10]). *Let $\mathcal{L}$ be a class of continuous, non-decreasing functions $\mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0}$ such that $\beta(\mathcal{L}) < 1$. The price of anarchy of any NCG with element latency functions drawn from $\mathcal{L}$ is no more than $\frac{1}{1 - \beta(\mathcal{L})}$.*

Proof. Let a be a Nash equilibrium and a^* an optimal action distribution. Then

$$\begin{aligned}
\mathsf{SC}(a) &\leq \mathsf{SC}^a(a^*) && \text{by (VAR)} \\
&= \mathsf{EC}^a(a^*) && \text{since } C = R \\
&= \sum_{e \in E} \big(\ell_e(g_e(a))g_e(a^*) - \ell_e(g_e(a^*))g_e(a^*)\big) + \mathsf{EC}(a^*) \\
&= \sum_{e \in E} (\ell_e(\underbrace{g_e(a)}_{v:=}) - \ell_e(\underbrace{g_e(a^*)}_{x:=}))\, \underbrace{g_e(a^*)}_{x=} + \mathsf{EC}(a^*)
\end{aligned}$$

$$
\begin{aligned}
&\le \sum_{e\in E} \beta(\mathcal{L})\ell_e(g_e(a))g_e(a) + \mathsf{EC}(a^*) && \text{by (7), see also (8)}\\
&= \beta(\mathcal{L})\mathsf{EC}(a) + \mathsf{EC}(a^*)\\
&= \beta(\mathcal{L})\mathsf{SC}(a) + \mathsf{SC}(a^*). && \text{since } C = R
\end{aligned}
$$

Hence $(1-\beta(\mathcal{L}))\mathsf{SC}(a) \le \mathsf{SC}(a^*)$, and so, using $\beta(\mathcal{L}) < 1$, we receive the claimed bound. □

We will now derive $\beta(\mathrm{Poly}^+(p))$ (and so $\alpha(\mathrm{Poly}^+(p))$) for some fixed $p \in \mathbb{N}_{\ge 1}$. We can restrict the analysis to the case that only element latency functions of the form $\ell_e(x) = c_e x^{q_e}$ are used, since we can always replace an element f with $\ell_f(x) = \sum_{q=0}^{p} c_{f,q}x^q$ by $p+1$ elements with element latency functions $x \mapsto c_{f,0}x^0$ to $x \mapsto c_{f,p}x^p$, respectively, yielding an equivalent NCG.

Let $\ell(x) = cx^q$, $c \ge 0$, $q \in \mathbb{N}_{\ge 1}$ (the case $q = 0$ trivially contributes a value of 0 to the supremum in (7)). Fix $v \ge 0$ and set $\phi_{\ell,v} : \mathbb{R}_{\ge 0} \longrightarrow \mathbb{R}, x \mapsto (\ell(v)-\ell(x))x$. Then, as noted before, $\phi_{\ell,v}(0) = \phi_{\ell,v}(v) = 0$, and, by monotonicity of ℓ, $\phi_{\ell,v}(x) \ge 0$ for all $x \le v$, and $\phi_{\ell,v}(x) \le 0$ for all $x \ge v$. Hence, there exists $x^* \in (0, v)$ such that $\phi_{\ell,v}(x^*) = \max_{x\ge 0} \phi_{\ell,v}(x)$. Real calculus yields $\phi'_{\ell,v}(x^*) = 0$, and hence $x^* = \frac{v}{(q+1)^{1/q}}$ and

$$
\begin{aligned}
\beta(\mathrm{Poly}^+(p)) &= \sup_{\substack{(x\mapsto cx^q)\in \mathrm{Poly}^+(p)\\ v\ge 0}} \frac{1}{cv^{q+1}}\phi_{(x\mapsto cx^q),v}\left(\frac{v}{(q+1)^{1/q}}\right)\\
&= \sup_{\substack{(x\mapsto cx^q)\in \mathrm{Poly}^+(p)\\ v\ge 0}} \frac{1}{cv^{q+1}}\left(cv^q - c\frac{v^q}{1+q}\right)\frac{v}{(q+1)^{1/q}}\\
&= \sup_{q\in[p]} \left(1-\frac{1}{1+q}\right)\frac{1}{(q+1)^{1/q}}\\
&= \sup_{q\in[p]} -\left(-\frac{1}{(q+1)^{1/q}} + \frac{1}{(1+q)^{(1+q)/q}}\right)\\
&= \sup_{q\in[p]} q(q+1)^{-(q+1)/q},
\end{aligned}
$$

where the last step follows as in (3). The maximum is attained for $q = p$, and so $\beta(\mathrm{Poly}^+(p)) = p(p+1)^{-(p+1)/p}$ and also (6). For illustration, a plot of $\alpha(\mathrm{Poly}^+(p))$ is shown in Fig. 2.

Good upper bounds on β can also be derived in a simpler way and for more general classes of functions than polynomials; see [10, Sec. 4] for details.

4.3 Computation of Optima and Nash Equilibria; The Potential

The set $\mathcal{A}$ is convex, and so is the function $\mathsf{SC} = \mathsf{EC}$, if, e.g., each $x \mapsto \ell_e(x)x$ is convex, as it is the case for standard element latency functions. Hence, for standard element latency functions, optima are characterized by the following convex program:

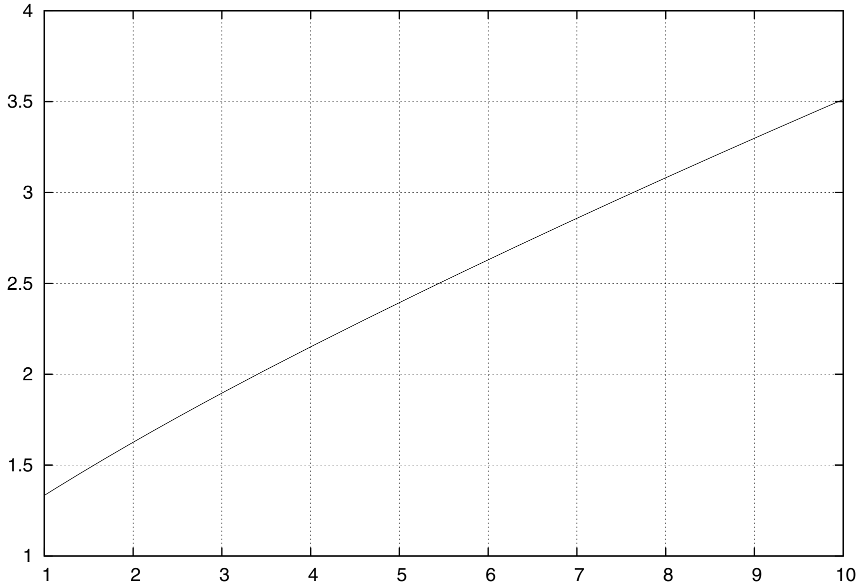


Fig. 2. The anarchy value $\alpha(\mathrm{Poly}^+(p))$ plotted over $p = 1, \ldots, 10$

$$\begin{aligned} \min \quad & \sum_{e \in E} \ell_e(x_e) x_e \\ \text{s.t.} \quad & x_e - \sum_{S \in \mathfrak{S}(e)} C_{eS} a_S = 0 \quad \forall\, e \in E \\ & d_i - \sum_{S \in \mathfrak{S}_i} a_S = 0 \quad \forall\, i \in [N] \\ & -a_S \leq 0 \quad \forall\, S \in \mathfrak{S} \end{aligned} \tag{OPT CP}$$

The constraints ensure that for any feasible pair $(x, a) = ((x_e)_{e \in E}, (a_S)_{S \in \mathfrak{S}})$ we have that $a \in \mathcal{A}$ and $x = \boldsymbol{g}(a)$. Convex programs can be solved in polynomial time up to an arbitrarily small error, under some mild additional assumptions. Some methods require certain boundedness conditions, smoothness and efficiently computable first and second order derivatives of the involved functions. See, e.g., [11] for a comprehensive treatment of convex optimization.

Interested in Nash equilibria, define $\hat{\ell}_e : \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0}, x \mapsto \int_0^x \ell_e(t) dt$ for each $e \in E$, and $\varPhi : \mathbb{R}^m_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0}, (x_e)_{e \in E} \mapsto \sum_{e \in E} \hat{\ell}(x_e)$. We call $\varPhi$ the *potential*; note that $\nabla \varPhi = \boldsymbol{\ell}$. Since each ℓ_e is non-decreasing, each $\hat{\ell}_e$ and the potential $\varPhi$ are all convex, and hence the following is a convex program (not only for standard element latency functions).

$$\begin{aligned}
\min \quad & \Phi((x_e)_{e\in E}) \quad (= \sum_{e\in E} \hat{\ell}_e(x_e)) \\
\text{s.t.} \quad & x_e - \sum_{S\in\mathfrak{S}(e)} C_{eS} a_S = 0 \quad \forall\, e \in E \\
& d_i - \sum_{S\in\mathfrak{S}_i} a_S = 0 \quad \forall\, i \in [N] \\
& -a_S \leq 0 \quad \forall\, S \in \mathfrak{S}
\end{aligned} \tag{Nash CP}$$

Optimal solutions to (Nash CP) coincide with Nash equilibria for the NCG with element latency functions $\boldsymbol{\ell}$. This has been noted by Beckmann et al. [12], Dafermos and Sparrow [13], and Braess [8,9]. We will give a proof below; again, it is crucial for the proof that $C = R$.

A drawback in terms of the practicability of (OPT CP) and (Nash CP) is that they involve as many variables as strategies, which might be exponential in the number of elements. However, in the case of unicast routing we can remedy this by using flow-conservation rules as constraints, hence expressing everything on edge-level.

Theorem 6. *A feasible* $(x, a) \in \mathbb{R}^{m+n}_{\geq 0}$ *is optimal for* (Nash CP) *if and only if the action distribution* a *is a Nash equilibrium.*

Proof. We give names to the constraints of (Nash CP): define $f_S(x, a) := -a_S$ for all $S \in \mathfrak{S}$, $h_i(x, a) := d_i - \sum_{S\in\mathfrak{S}_i} a_S$ for all $i \in [N]$, and $h_e(x, a) := x_e - \sum_{S\in\mathfrak{S}(e)} C_{eS} a_S$ for all $e \in E$. Like the constraints, we will also consider Φ a function of $m + n$ variables, though it only depends on the first m. Slater's condition (see, e.g., [11, Sec. 5.2.3]) is easily verified: take $a_S := \frac{d_i}{n_i}$ for all $S \in \mathfrak{S}_i$ and $i \in [N]$, and $x_e := g_e(a)$ for all $e \in E$, which is > 0 since C has no row of only zeros. Then (x, a) is in the interior of the domain of (Nash CP), i.e., the set where the objective function and all constraint functions are defined, which is $\mathbb{R}^n_{\geq 0}$ here.

Hence a feasible (x, a) is optimal if and only if the Karush-Kuhn-Tucker (KKT) conditions hold. These conditions are (see, e.g., [11, Sec. 5.5.3]): there exist real numbers $(\lambda_S)_{S\in\mathfrak{S}}, (\nu_i)_{i\in[N]}, (\nu_e)_{e\in E}$ such that $\lambda_S \geq 0$ and $\lambda_S a_S = 0$ for all $S \in \mathfrak{S}$ and

$$\nabla\Phi(x, a) + \sum_{S\in\mathfrak{S}} \lambda_S \nabla f_S(x, a) + \sum_{i\in[N]} \nu_i \nabla h_i(x, a) + \sum_{e\in E} \nu_e \nabla h_e(x, a) = 0.$$

This last equation is equivalent to

$$\begin{aligned}
\ell_e(x_e) + \nu_e = 0 \quad & \forall e \in E \\
-\lambda_S - \nu_i - \sum_{e\in S} \nu_e C_{eS} = 0 \quad & \forall S \in \mathfrak{S}_i \quad \forall i \in [N].
\end{aligned}$$

It follows that a feasible (x, a) is optimal if and only if there exist $(\lambda_S)_{S \in \mathfrak{S}}$, $(\nu_i)_{i \in [N]}$ such that

$$\begin{aligned} \lambda_S \geq 0 \text{ and } \lambda_S a_S = 0 \quad & \forall S \in \mathfrak{S} \\ \sum_{e \in S} \ell_e(x_e) C_{eS} - \nu_i = \lambda_S \quad & \forall S \in \mathfrak{S}_i \quad \forall i \in [N]. \end{aligned}$$

Since $C = R$, we have $\sum_{e \in S} \ell_e(x_e) C_{eS} = L_S(a)$. Reformulating further, we see that a feasible (x, a) is optimal if and only if there exist $(\nu_i)_{i \in [N]}$ such that we have for all $i \in [N]$ and $S \in \mathfrak{S}_i$ the following: $L_S(a) - \nu_i \geq 0$, and $L_S(a) - \nu_i = 0$ in case of $a_S > 0$. This is the Nash condition (we have $\nu_i = \Lambda_i(a)$ if it holds). □

The implications of this result go beyond the computation of equilibria. Assume a standard class of element latency functions and recall the definition of the marginal cost function ℓ_e^* for element e, which is the derivative of $x \mapsto \ell_e(x)x$, i.e., $\ell_e^*(x) = \ell'(x)x + \ell(x)$. Let for a NCG Γ denote Γ^* a modification of Γ where each ℓ_e is replaced by ℓ_e^*. Since $\int_0^x \ell_e^*(t)dt = \ell_e(x)x$, the potential for Γ^* coincides with EC and so with SC for Γ. It follows from Thm. 6 that an action distribution is a Nash equilibrium for Γ^* if and only if it is optimal for Γ. This is part of the motivation behind the techniques in the proof of Thm. 4; more on this is explained in detail (for unicast routing) in [7, Sec. 3.2 and 3.3].

5 Our Results for NCRCGs

Most proofs in this section are omitted and can be found in [14].

5.1 Multicast Routing and Lower Bound

The initial motivation for the NCRCG model comes from multicast routing. In multicast routing, we are given N sources s_i, $i \in [N]$, each of them associated with k_i sinks $t_i^1, \ldots, t_i^{k_i}$, where k_i is some positive natural number. For each $i \in [N]$, a demand of d_i has to be routed from s_i to each sink $t_i^1, \ldots, t_i^{k_i}$ simultaneously. Again, we assume the demand to represent a continuum $[0, d_i]$ of players. The underlying structure, as in unicast routing, is a directed multigraph (V, E) where each edge $e \in E$ has a latency function ℓ_e.

Let us fix one $i \in N$. To realize the desired routing for some fraction of the demand, we have to choose k_i paths $\mathcal{P} := \{P_1, \ldots, P_{k_i}\}$, where P_j connects s_i with t_i^j for each $j \in [k_i]$. Such a set $\mathcal{P}$ will constitute a strategy; denote for later $\mathcal{P}(e) := \{P \in \mathcal{P};\ e \in P\}$ for each edge e. Hence, to model multicast as an NCRCG, matrices C and R have to somehow represent all eligible choices of such collections of paths.

Before we can write out C and R, further decisions have to be made. There are at least two ways to *realize* a flow using a strategy $\mathcal{P}$. One is to realize a flow in the usual sense: the amount of flow entering a node is exactly the amount of flow leaving the node. This will be called a *conservation flow.* When using conservation flows, the consumption number of an edge e regarding strategy $\mathcal{P}$ is $|\mathcal{P}(e)|$, i.e., the number of sinks served via e. A smarter way to realize $\mathcal{P}$ is to exploit the fact that we deal with *data* to be routed. Unlike material goods or physical flows, data can be duplicated virtually without cost, provided that the nodes in the network offer such a feature. Thus the same data has to be sent down an edge only *once*, no matter how many sinks are served by this. We call this *duplication flows*. When using duplication flows, the consumption number of an edge regarding $\mathcal{P}$ is 1 if it is contained in some path from $\mathcal{P}$, and 0 otherwise.

We now consider relevance numbers. There are many different reasonable ways in which the latency of a collection of paths $\mathcal{P}$, i.e., the strategy latency, could be understood; we point out four of them in [15]. Two interesting ones can be modeled by assigning relevance numbers to edges. In the first model, we would simply sum up the edge latencies of all edges involved, hence the relevance number of an edge is 1 if that edge is contained in some path from $\mathcal{P}$, and 0 otherwise. We call this *edge-based strategy latency*. In the other model, we consider an edge more relevant if it serves many sinks. We will set its relevance number to $|\mathcal{P}(e)|$ for that model[5] and refer to this model by the term *path-based strategy latency*, since it is equivalent to taking the sum of the latencies of all paths in $\mathcal{P}$, where the latency of a path is the sum of the latencies of the edges (exactly as in unicast routing).

Combining conservation flows with path-based strategy latency, or combining duplication flows with edge-based strategy latency, yields $C = R$, and is hence covered by previous analysis. However, combining conservation flows with edge-based latency, or combining duplication flows with path-based strategy latency, yields $C \neq R$ in general. We will focus on the latter combination in the following.

Consider the instance shown in Fig. 3. We have one player class with demand $d = 1$ and two sinks t^1 and t^2. There are three strategies as depicted in Fig. 4. For example, in strategy 1, flow is routed to sink t^1 (serving that sink) and then duplicated and sent to sink t^2. We give matrices C and R. Recall that columns correspond to strategies and rows correspond to elements, i.e., edges. Thus

$$C = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad \text{and} \quad R = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} .$$

A Nash equilibrium is given if all players choose strategy 3, i.e., the action distribution $a := (0, 0, 1)$. This can be checked easily, and we only point out the basic principle here: all players experience latency $1 + 1 = 2$ under a. Assume that a player considers switching from strategy 3 to, say, strategy 1. Edge 3 has element latency 0, however, the relevance number of edge 1 increases from 1 to 2 when switching strategies, and so that player would still experience latency 2. Hence there is no incentive for such a switch.

[5] Many other expressions involving $|\mathcal{P}(e)|$ would also be reasonable, e.g., $|\mathcal{P}(e)|^c$ for some $c > 0$. All such variations can be expressed in the NCRCG model.

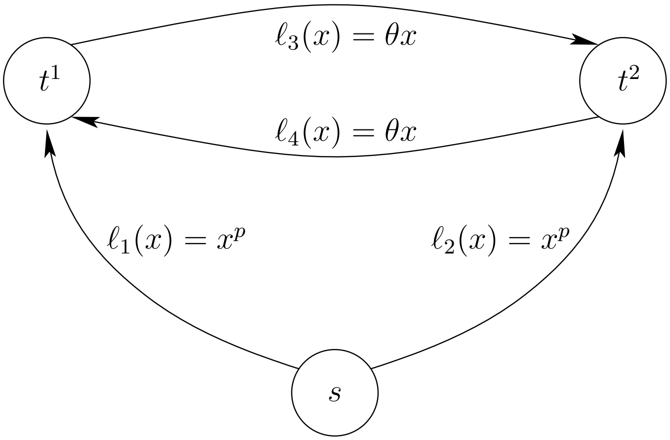
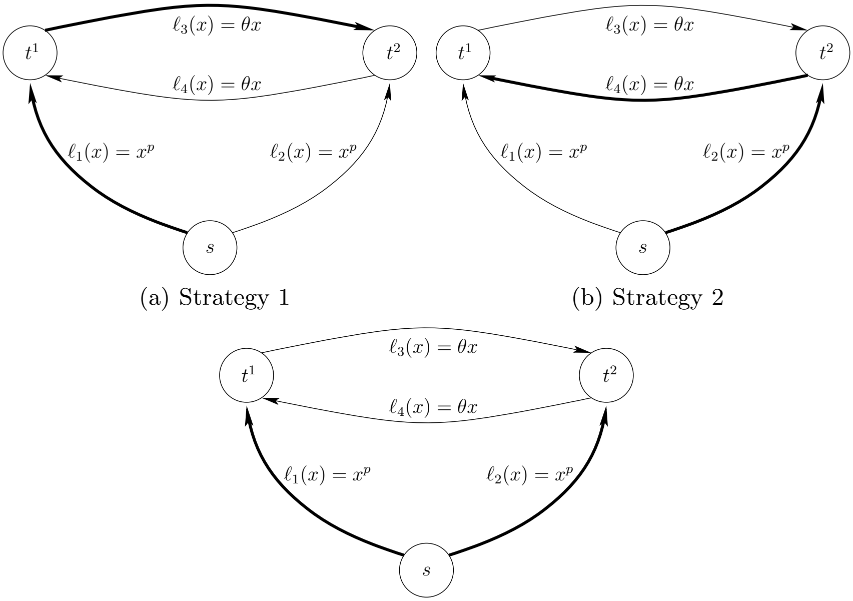


Fig. 3. Multicast example with parameters $p \in \mathbb{N}_{\geq 1}$ and $\theta \geq 0$



(a) Strategy 1

(b) Strategy 2

(c) Strategy 3

Fig. 4. The three strategies

For comparison, consider the action distribution $a^* := (\frac{1}{2}, \frac{1}{2}, 0)$. It has much better social cost, providing a lower bound on the price of anarchy

$$\frac{\mathsf{SC}(a)}{\mathsf{SC}(a^*)} = \frac{2}{2(2\frac{1}{2^p} + \theta\frac{1}{2})\frac{1}{2}} = \frac{1}{\frac{1}{2^p} + \theta\frac{1}{4}} = \frac{2^p}{1 + \theta\frac{1}{4}2^p} = \Omega(2^p) \quad \text{for } \theta = O(2^{-p}).$$

This clearly breaks the anarchy-value bound, which would have been $O(\frac{p}{\ln p})$. The example can be generalized to any number k of sinks, yielding a price of anarchy of $\Omega(k^p)$, see [15] for details.

Another interesting observation is that for the case of $\theta = 0$ the action distribution a^* is also a Nash equilibrium, resulting in a gap of $\Omega(2^p)$ between price of stability and price of anarchy. This exponential gap is another fundamental difference to the case $C = R$, where these two quantities coincide.

5.2 Lower and Upper Bound for Polynomials

We turn back to NCRCGs in general and first wish to provide a worst-case lower bound on the price of anarchy when element latency functions are drawn from $\text{Poly}^+(p)$, for some $p \in \mathbb{N}$. We need a measure for the deviation of C from R. We introduce the following two parameters:

$$\gamma_1 := \max\{\frac{C_{e,S}}{R_{e,S}};\ S \in \mathfrak{S}, e \in S\},$$
$$\gamma_2 := \max\{\frac{R_{e,S}}{C_{e,S}};\ S \in \mathfrak{S}, e \in S\}.$$

Recall that a strategy S interpreted as a set $S \subseteq E$ means $S = \{e \in E;\ C_{eS} \neq 0\}$, which by our requirements on C and R is a non-empty set and the same as the set $\{e \in E;\ R_{eS} \neq 0\}$. Hence these parameters are well-defined. We have $\gamma_1 = \gamma_2 = 1$ in case of $C = R$.

The previous section immediately gives a worst-case lower bound of $\Omega(\gamma_2^p)$ with $\gamma_2 = 2$; using the extended construction from [15] even for arbitrary integers γ_2. We can even generalize further.

Theorem 7. *Let $c, r \in \mathbb{R}_{\geq 1}$. There exist NCRCGs with element latency functions only from* $\text{Poly}^+(p)$ *with $\gamma_1 = c$ and $\gamma_2 = r$ such that the price of anarchy is at least $(\gamma_1\gamma_2)^p$.*

For an upper bound, we consider NCRCGs with element latency functions drawn from $\text{Poly}^+(p)$ for some fixed $p \in \mathbb{N}$. The main obstacle for upper bounds on the price of anarchy is that in general (mixed) social cost differs from (mixed) total element cost. However, we still have characterization (VAR) for Nash equilibria and the basic idea from Thm. 5. We can use this to prove an upper bound matching that of Thm. 7 up to a factor of $\gamma_1\gamma_2$ or $\alpha(\text{Poly}^+(p))\gamma_1\gamma_2$, depending on the range of $\gamma_1\gamma_2$.

Theorem 8. *The price of anarchy in an NCRCG with element latency functions drawn from* $\text{Poly}^+(p)$ *is no more than*

$$\begin{cases} (\gamma_1\gamma_2)^{p+1} & \text{if } \gamma_1\gamma_2 \geq (1+p)^{\frac{1}{p}} \\ \alpha \cdot (\gamma_1\gamma_2)^{p+1} & \text{if } \gamma_1\gamma_2 \leq (1+p)^{\frac{1}{p}} \end{cases}, \quad \text{where } \alpha = \alpha(\text{Poly}^+(p)) = \Theta(\frac{p}{\ln p}).$$

5.3 Computation

Recall that in the case $C = R$ with standard element latency functions, the social cost $\mathsf{SC} = \mathsf{EC}$ was a convex function (even a separable one if considered a function of the congestions), and so optima were characterized by a convex program. In the general case, however, SC is not always convex. Let us assume that SC can be extended to a twice differentiable function on some open convex set $U \subseteq \mathbb{R}^n$ with $\mathcal{A} \subset U$. This is the case, e.g., for polynomial element latency functions, where we can take $U := \mathbb{R}^n$. Then SC is convex on $\mathcal{A}$ if the Hessian $\nabla^2 \mathsf{SC}(v)$ is positive semidefinite for all $v \in V$ for some open convex set V with $\mathcal{A} \subset V \subseteq U$. Since $\mathcal{A}$ is not an open set, the converse does not hold. (It would suffice to consider open convex sets V such that $\mathcal{A} \subseteq \overline{V}$. However, since $\mathcal{A}$ has empty interior, this gives no advantage.)

Consider the case of affine element latency functions. Let for each $e \in E$ be $\ell_e(x) = \theta_e x + \tau_e$ with $\theta_e, \tau_e \in \mathbb{R}_{\geq 0}$. Then SC has constant Hessian, i.e., $\nabla^2 \mathsf{SC}(v)$ does not depend on $v \in \mathbb{R}^n$ but only on C, R, and $(\theta_e)_{e \in E}$. We have for all $v \in \mathbb{R}^n$

$$\nabla^2 \mathsf{SC}(v) = \left(\sum_{e \in E} \theta_e (R_{eS_1} C_{eS_2} + R_{eS_2} C_{eS_1}) \right)_{\substack{S_1 \in \mathfrak{S} \\ S_2 \in \mathfrak{S}}} .$$

Hence it can be checked whether $\nabla^2 \mathsf{SC}$ is positive semidefinite on some open convex set $V \supset \mathcal{A}$, we only have to check *one* matrix for being positive semidefinite (e.g., by inspecting the eigenvalues). No concrete choice for V has to be made – the Hessian is the same in each point of $\mathbb{R}^n$. A check for non-convexity based on this can deliver false positives, but it cannot deliver false negatives. We summarize the three cases that can occur for general C and R:

1. SC may be a convex function. This is the case for the multicast instance in Fig. 3, for $p = 1$ and $\theta = 1$, as can simply be checked by showing that
$$\nabla^2 \mathsf{SC}(\cdot) = \begin{pmatrix} 6 & 0 & 3 \\ 0 & 6 & 3 \\ 3 & 3 & 4 \end{pmatrix}$$
is positive semidefinite; its spectrum consists of $5 \pm \sqrt{19}$ and 6.
2. SC may have a Hessian that is not positive semidefinite on any open set $V \supset \mathcal{A}$, but yet SC is convex on $\mathcal{A}$. That is possible since $\mathcal{A}$ is not an open set. As an example, consider $C := \begin{pmatrix} 1 & 1 \end{pmatrix}$ and $R := \begin{pmatrix} 3 & 1 \end{pmatrix}$, let the one element have latency function $x \mapsto x$, and a demand $d = 1$ be given. Then, SC is convex on $\mathcal{A} = \{(a_1, a_2);\ a_1 + a_2 = 1\}$. However $\nabla^2 \mathsf{SC}(\cdot) = \begin{pmatrix} 6 & 4 \\ 4 & 2 \end{pmatrix}$ is not positive semidefinite; it has got the negative eigenvalue $4 - \sqrt{20}$.
3. SC may be non-convex on $\mathcal{A}$, as can be seen easily by modifying the above example to $C := \begin{pmatrix} 1 & 2 \end{pmatrix}$.

It is yet possible to give a full characterization of the convexity of SC, namely via the projected Hessian. For simplicity of notation, we restrict ourselves to the case of $N = 1$. Define

$$T : \mathbb{R}^{n-1} \longrightarrow \mathbb{R}^n, (v_1, \ldots, v_{n-1}) \mapsto \underbrace{\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ -1 & -1 & \ldots & -1 \end{pmatrix}}_{M_T :=} v + \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ d \end{pmatrix}.$$

Then T is an affine mapping between $\mathbb{R}^{n-1}$ and the hyperplane in $\mathbb{R}^n$ that contains $\mathcal{A}$. We can show that SC is convex on $\mathcal{A}$ if and only if the projected Hessian $H'(v) := M_T^{\perp} \cdot \nabla^2 \mathsf{SC}(v) \cdot M_T$ is positive semidefinite on

$$\mathcal{A}' := \{(v_1, \ldots, v_{n-1});\ v_i > 0\, \forall i \in [n-1] \text{ and } \sum_{i=1}^{n-1} v_i < d\}.$$

In case of affine element latency functions, $\nabla^2 \mathsf{SC}$ is constant, and so is H'. Hence a check for H' being positive semidefinite is practical as described above. The generalization to $N > 1$ is straight-forward. We summarize the discussion in the following theorem.

Theorem 9. *For an NCRCG with affine element latency functions we can efficiently determine whether* SC *is a convex function or not.*[6]

If SC is convex, then the following program characterizing optimal action distributions is a convex one:

$$\begin{aligned} &\min\ \mathsf{SC}(a) \\ &\text{s.t. } a \in \mathcal{A} \end{aligned} \qquad \text{(OPT NLP)}$$

For the computation of Nash equilibria, recall that for the case $C = R$ Nash equilibria are the optimal solutions to the convex program (Nash CP) on page 307. This result is based on the KKT theorem and seems to depend crucially on $C = R$. Hence we do not know how to apply it in the general case. We can, however, still characterize Nash equilibria by a minimization problem.

$$\begin{aligned} &\min\ \mathsf{SC}(a) - \lambda \cdot d \\ &\text{s.t. } a \in \mathcal{A} \\ &\qquad \lambda \in \mathbb{R}^N \\ &\qquad \lambda_i \leq L_S(a) \quad \forall S \in \mathfrak{S}_i\, \forall i \in [N] \end{aligned} \qquad \text{(Nash NLP)}$$

We can show that the optimum value of that non-linear program is 0 and that the set of optima coincides with the set of Nash equilibria. The constraints of that program, however, are not necessarily convex anymore, since one of the constraints is ‘$\lambda_i - L_S(a) \leq 0$’, and $-L_S(\cdot)$ is not always a convex function.

[6] Up to numerical inaccuracies involved when checking a matrix for being positive semidefinite, e.g., via computation of eigenvalues.

On the upside, we can very easily check a solution returned by some NLP solver whether it in fact is a Nash equilibrium – just see whether condition (1) holds. This check also helps in the case of non-convex SC. If all element latency functions are affine, (Nash NLP) even has got only linear, hence convex constraints.

In order to compute the price of anarchy or stability, worst and best Nash equilibria need to be computed. We can force the computed equilibrium (if any) to have a certain social cost by an additional linear constraint:

$$c_0 \leq \lambda \cdot d \leq c_1. \tag{10}$$

By successively increasing c_0 (decreasing c_1), or by binary search, worst (best) Nash equilibria can be found, up to some arbitrarily small error – provided that we can solve (Nash NLP) together with (10) optimally or arbitrarily close to optimality. That is the case, in particular, for affine element latency functions and convex SC, since then (Nash NLP) together with (10) is a linearly constrained convex *quadratic* program, or can be reformulated as such using the transformation T.

The interval to be searched is $[\mathsf{OPT}, \tilde{\rho}\,\mathsf{OPT}]$, where $\tilde{\rho}$ denotes an upper bound on the price of anarchy. Hence, for an error[7] of at most ε, a binary search needs $\lceil \log_2(\frac{1}{\varepsilon}(\tilde{\rho}-1)\mathsf{OPT})\rceil$ steps. For the experimental series briefly sketched in the next section, we however used a successive increase of c_0 by a small increment, which we found to be practical. Using $\tilde{\rho} = \frac{4}{3}(\gamma_1\gamma_2)^2$ for affine element latency functions, we have:

Theorem 10. *Let Γ be an NCRCG with affine element latency functions and convex SC. We can compute worst and best Nash equilibria of Γ up to an error of ε (and an additional error introduced by convex programming) by solving one instance of* (OPT NLP)*, and by solving or showing to be infeasible at most $\lceil \log_2(\frac{1}{\varepsilon}(\frac{4}{3}(\gamma_1\gamma_2)^2-1)\mathsf{OPT})\rceil$ instances of* (Nash NLP) *with* (10)*, which all can be formulated as linearly constrained convex quadratic programs.*

Practicability may depend on the number of variables – and so for our programs in particular on the number of strategies. If, e.g., the set of strategies was to enumerate *all* possible multicast path collections (or trees) in a network, this could be exponential in the number of links. Efficiently handling such applications is future work.

5.4 Experimental Results and Conjecture

We designed and wrote a clean implementation that solves (OPT NLP) and (Nash NLP) with (10), using existing NLP solvers [16,17] as back-ends. It can solve specific instances as well as generate and solve series of random instances. We conducted extensive experimental studies of the price of anarchy in NCRCGs. It is beyond the scope of this article to describe the experiments or the

[7] Meaning that the Nash equilibrium found shall have social cost at least $\mathsf{SC}(a) - \varepsilon$ or at most $\mathsf{SC}(a) + \varepsilon$, where a is a worst or a best Nash equilibrium, respectively, depending on whether we wish to approximate a worst or a best one.

deployed random model in detail, and so we refer to [14,18] for all details instead. Experiments were done on randomly generated instances with polynomial element latency functions. Effort was put into creating a *variety* of different kinds of instances. Although we treated several million instances with $p \in \{1,2\}$ and several hundred thousand with $p = 3$, for none of them with $\gamma_1\gamma_2 \geq (1+p)^{\frac{1}{p}}$ we could prove a price of anarchy exceeding that of Thm. 7 by more than 1%. Attributing the 1% to numerical inaccuracies, we make the following conjecture.

Conjecture. The price of anarchy in an NCRCG with element latency functions drawn from $\mathrm{Poly}^+(p)$ is no more than

$$\begin{cases} (\gamma_1\gamma_2)^p & \text{if } \gamma_1\gamma_2 \geq (1+p)^{\frac{1}{p}} \\ \alpha \cdot (\gamma_1\gamma_2)^p & \text{if } \gamma_1\gamma_2 \leq (1+p)^{\frac{1}{p}} \end{cases}, \quad \text{where } \alpha = \alpha(\mathrm{Poly}^+(p)) = \Theta(\frac{p}{\ln p}).$$

Fig. 5 shows results for some random instances with $p = 1$. A dot is drawn for each instance and positioned horizontally according to its bound given by the conjecture[8] and positioned vertically according to the observed price of anarchy. A line marks the conjectured bound. A dot that was clearly positioned to the upper left of that line would have constituted a counter-example to the conjecture. There are no such dots; no instance exceeds the conjecture by more than 1%. Some dots are positioned vertically below 1, which means that the optimum was not found, and a Nash equilibrium had lower social cost than the false optimum. These dropout dots disappear, as expected, when we remove all instances with non-convex SC from the data set.

We speak of *observed price of anarchy* here meaning the value $\mathsf{SC}(a)/\mathsf{SC}(a^*)$, where a is the Nash equilibrium with highest social cost which we could find by means of trying to solve (Nash NLP) with (10) and successively increasing c_0, and a^* is the action distribution with the smallest social cost which we could find by means of trying to solve (OPT NLP). The observed price of anarchy is a lower bound on the price of anarchy; in case of $p = 1$ and convex SC, it equals the price of anarchy (up to numerical errors).

The random instances presented in Fig. 5 were generated as follows: The number of elements was fixed to $m = 4$, the demand to $d = 1$ (we used only one player class), the number of strategies ranges in $n \in \{2,3,4,5,6,9\}$, consumption numbers were drawn from the interval $[1.0, 9.0]$, and relevance numbers were restricted to different intervals in different series of the experiment: $[1.0, 1.0]$, $[1.0, 5.0]$, or $[1.0, 9.0]$. Choosing the most restrictive range for relevance numbers, i.e., $[1.0, 1.0]$, has shown to generate the most instances with an observed price of anarchy close to or on the conjectured bound. In half of the experiments, element latency functions of the form $\ell(x) = \sum_{k=0}^{p} \theta_k x^k$ with random coefficients $\theta_0, \ldots, \theta_p$ were used, and for the other half, simply $\ell(x) = x^p$ was used.

Closer examination of Fig. 5 shows a peculiarity for small $\gamma_1\gamma_2 \neq 1$. It appears as if the conjectured bound is still too pessimistic in this range, and indeed, guided by the experimental results, we were able to refine the bound for small

[8] We use a value of $\gamma_1\gamma_2$ that was optimized by scaling C and R, as explained in [14].

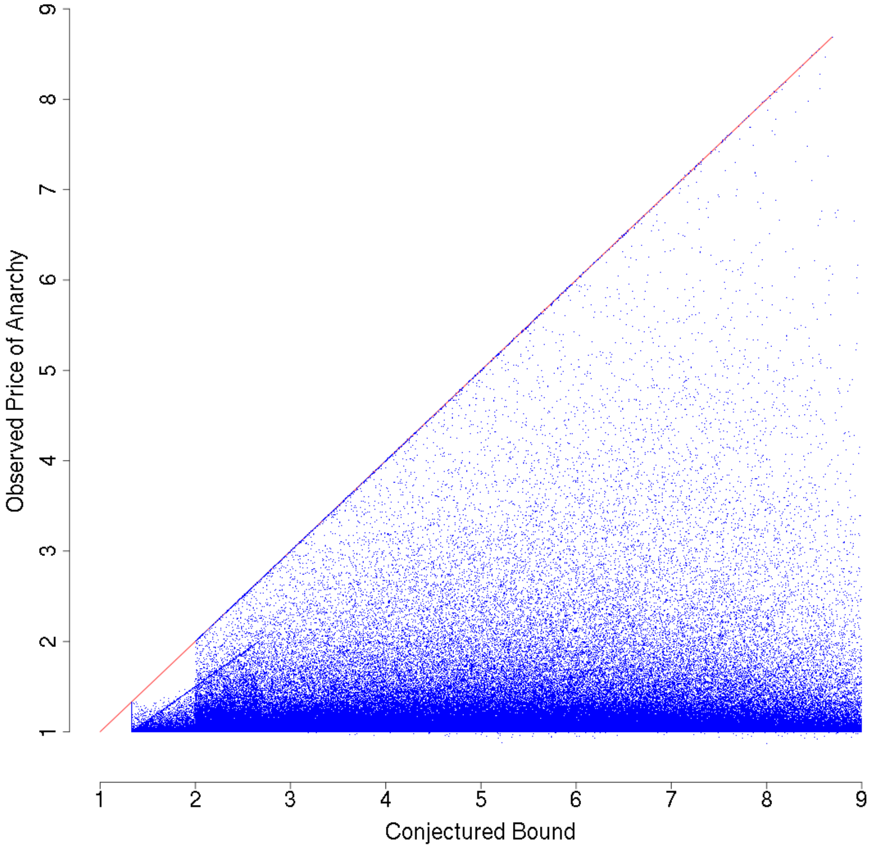


Fig. 5. Scatter plot for random instances with affine element latency functions, i.e., $p = 1$. A dot is drawn for each instance and positioned horizontally according to its bound given by the conjecture and positioned vertically according to the observed price of anarchy. A line marks the conjectured bound. Only those with $\gamma_1\gamma_2$ smaller than the maximum observed price of anarchy were selected from the data set, for the sake of a better horizontal scale, resulting in approximately 650,000 instances shown out of a total of 864,000. Note that several dots are close to or on the line, but none above it. Plots and tables for larger data sets are given in [18].

$\gamma_1\gamma_2$ in the conjecture as well as in the proven bound. Details are omitted here and are given in [14].

6 Further Topics

We completely left out at least two important topics in this survey. We name them here for completeness and give some references. The first concerns methods to reduce the price of anarchy. Available methods roughly fall into three categories:

1. Network design: detect edges in a network such that their removal from the network leads to better equilibria. As the name says, this has been studied in the context of network routing, and unicast in particular, see [7, Ch. 5] for a comprehensive treatment and the references therein for further background.
2. Stackelberg routing: centrally control some fraction of the demand and let the rest of the demand behave as usual. The hope is that this leads to better equilibria. Stackelberg routing has been studied for different unicast models, see [19], [7, Ch. 6], and also the references in the latter for further background.
3. Taxation: charge players depending on which strategies they use, assuming that they will try to minimize a linear combination of those taxes and latency. The hope is that this induces better equilibria. That idea was already discussed in [2].

   A positive result for unicast routing has recently been provided [20]. A straight-forward generalization to NCRCGs is possible under the additional requirement that all Nash equilibria have the same social cost, which is not the case in general (not even for multicast with strictly increasing edge latency functions [14]).

The second topic concerns dynamic processes which are usually based on some form of improvement steps taken by the players. There are several recent advances in that direction [21,22,23].

Acknowledgments. We thank our student Ole Kliemann for his work on the implementation for the computation of Nash equilibria. We thank Andreas Baltz and our student Sandro Esquivel for their contributions in early stages of the project. And we sincerely thank the Priority Program 1126 “Algorithmics of Large and Complex Networks” of the Deutsche Forschungsgemeinschaft for their financial support that made this work possible.

References

1. Roughgarden, T., Tardos, É.: Bounding the inefficiency of equilibria in nonatomic congestion games. Games and Economic Behavior 47(2), 389–403 (2004)
2. Pigou, A.C.: The economics of welfare. Macmillan, London (1920)
3. Wardrop, J.G.: Some theoretical aspects of road traffic research. Proceedings of the Institution of Civil Engineers, Part II 1, 325–362 (1952)
4. Smith, M.J.: The existence, uniqueness and stability of traffic equilibria. Transportation Research Part B: Methodological 13(4), 295–304 (1979)
5. Nagurney, A.: Network Economics: A Variational Inequality Approach. Springer, Heidelberg (1999)
6. Kinderlehrer, D., Stampacchia, G.: An Introduction to Variational Inequalities and Applications. Academic Press, New York (1980)
7. Roughgarden, T.: Selfish routing. Ph.D thesis, Cornell University, Ithaca, NY, USA (2002)

8. Braess, D.: Über ein Paradoxon aus der Verkehrsplanung. Unternehmensforschung 12, 258–268 (1968)
9. Braess, D., Nagurney, A., Wakolbinger, T.: On a paradox of traffic planning. Transportation Science 39, 446–450 (2005); Translation from the original German
10. Correa, J.R., Schulz, A.S., Stier Moses, N.E.: Selfish routing in capacitated networks. Mathematics of Operations Research 29(4), 961–976 (2004)
11. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
12. Beckmann, M., McGuire, C.B., Winsten, C.B.: Studies in the Economics and Transportation. Yale University Press, Haven (1956)
13. Dafermos, S.C., Sparrow, F.T.: The traffic assignment problem for a general network. Journal of Research of the National Bureau of Standards 73B(2), 91–118 (1969)
14. Kliemann, L.: The price of anarchy and computation of equilibria in non-atomic consumption-relevance congestion games. Technical Report 0814, Institut für Informatik, Christian-Albrechts-Universität Kiel (2008)
15. Baltz, A., Esquivel, S., Kliemann, L., Srivastav, A.: The price of anarchy in selfish multicast routing. In: Erlebach, T. (ed.) CAAN 2006. LNCS, vol. 4235, pp. 5–18. Springer, Heidelberg (2006)
16. Conn, A.R., Gould, N.I.M., Toint, P.L.: Lancelot: A FORTRAN Package for Large-Scale Nonlinear Optimization (Release A). Springer, New York (1992)
17. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming 106(1), 25–57 (2006)
18. Kliemann, L.: Experimental studies of the price of anarchy in consumption-relevance congestion games. Technical report, Institut für Informatik, Christian-Albrechts-Universität Kiel (2008) (in preparation)
19. Korilis, Y.A., Lazar, A.A., Orda, A.: Achieving network optima using Stackelberg routing strategies. IEEE/ACM Transactions on Networking 5(1), 161–173 (1997)
20. Fleischer, L., Jain, K., Mahdian, M.: Tolls for heterogeneous selfish users in multicommodity networks and generalized congestion games. In: FOCS, pp. 277–285. IEEE Computer Society, Los Alamitos (2004)
21. Fischer, S., Räcke, H., Vöcking, B.: Fast convergence to Wardrop equilibria by adaptive sampling methods. In: Kleinberg, J.M. (ed.) STOC, pp. 653–662. ACM, New York (2006)
22. Fischer, S., Olbrich, L., Vöcking, B.: Approximating Wardrop equilibria with finitely many agents. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 238–252. Springer, Heidelberg (2007)
23. Chien, S., Sinclair, A.: Convergence to approximate Nash equilibria in congestion games. In: Bansal, N., Pruhs, K., Stein, C. (eds.) SODA, pp. 169–178. SIAM, Philadelphia (2007)

New Data Structures for IP Lookup and Conflict Detection

Christine Maindorfer, Tobias Lauer, and Thomas Ottmann

Institut für Informatik
Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee, Gebäude 51
D-79110 Freiburg, Germany

Abstract. In this paper we present a survey of new data structures for the representation of dynamic range router tables that employ most specific range matching. We present the min-augmented range tree with a relaxed balancing scheme, allowing updates and rebalancing tasks to be decoupled. Utilizing this scheme, IP lookups are not as much delayed as in a strict balancing scheme. Furthermore, we outline additional improvements to an existing online conflict detection approach, saving space and update costs. For offline conflict detection and resolution we describe an efficient sweepline algorithm.

1 Introduction

The Internet is a global web of interconnected autonomous networks, a "network of networks", interconnected with routers. Each network, or Autonomous System (AS), is managed by different authorities and contains its own internal network of routers and subnetworks. Modern IP routers provide policy-based routing (PBR) mechanisms which complement the existing destination-based forwarding scheme. PBR requires network routers to examine multiple fields of the packet header in order to categorize them into "flows". The process of categorizing packets into flows in an Internet router is called packet classification. The function of the packet classification system is to match packet headers against a set of predefined filters. The relevant packet header fields include source and destination IP addresses, source and destination port numbers, protocol and others. Formally, a filter set consists of a finite set of n filters, $f_1, f_2 \dots f_n$. Each filter is a combination of d header field specifications, $H_1, H_2 \dots H_d$. Each header field specifies one of four kinds of matches: exact match, prefix match, range match, or masked-bitmap match. A packet p is said to match a filter f_i if and only if the fields in f_i match the corresponding header fields $H_1, H_2 \dots H_d$ in the specified way. Each filter f_i has an associated action that determines how a packet p is handled if p matches f_i. In the simplest case of destination-based forwarding, each filter consists of a single field, which is a destination prefix; the associated action is the next hop for packets matching the prefix. The task of resolving the next hop for an incoming packet is referred to as IP lookup. A route lookup

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 319–329, 2009.

requires finding the longest matching prefix among all matching prefixes for the given destination address.

Routers in different ASs use the Border Gateway Protocol (BGP) to exchange network reachability information. After applying local policies, a BGP router selects a single best route and advertises it to other BGP routers within the same AS. A router sends an announcement to notify its neighbors of a new route to the destination and sends a withdrawal to revoke the route when it is no longer available. From these updates an IP router constructs a forwarding table which contains a set of network addresses and the corresponding output link for packets destined for each network. There are two strategies to handle table updates. The first employs two copies of the table. Lookups are done on the *working* table, updates are performed on a *shadow* table. Periodically, the shadow table replaces the working table. In this mode of operation, packets may be misclassified. The amount of misclassifications depends on the periodicity with which the working table is replaced by an updated shadow. Further, additional memory is required for the shadow table. The second strategy performs updates directly on the working table. Here, no packet is improperly forwarded. However, IP lookup may be delayed while a preceding update completes. Since the performance of the lookup device plays a crucial role in the overall performance of the Internet, it is important that lookup and update operations are performed as fast as possible.

1.1 Preliminaries

In one dimension, a filter f can be regarded as a range $[u, v]$ on the number line. An address d matches f if $d \in [u, v]$. Usually, one-dimensional IP filters are specified as address prefixes. In geometric terms, a prefix $b_1 \dots b_k*$ can be mapped to a range in the form of $[b_1 \dots b_k 0 \dots 0, b_1 \dots b_k 1 \dots 1]$. For example, if the prefix length is limited by 5, 0010$*$ is represented by $[4, 5]$. An incoming packet with destination address $d = b_1 \dots b_w$ can be mapped to a point $p \in U$, where $U = [0, 2^w - 1]$ and $w = 32$ for IPv4 and $w = 128$ for IPv6. The longest matching prefix corresponds to the *most-specific* range of all ranges that contain the query point.

Definition 1. *Filter f_1 is more specific than filter f_2 iff $f_1 \subset f_2$. If two filters partially overlap, neither is more specific than the other.*

In one dimension, a set of prefix ranges has the property that any two ranges are either disjoint or one is completely contained in the other. Hence, for each query point d, there is a unique defined most-specific range that contains d, provided that the default filter spanning the entire universe U is included in the set. A query for the longest matching prefix for a given address thus corresponds to a stabbing query, where for a given point d the most specific range containing d has to be found. It has been observed that stabbing queries for sets of ranges on the line can be translated to range queries for so called south-grounded, semi-infinite ranges of points in the plane [1]. This is done by mapping each range $[u, v]$ to the point (v, u). Note that all such points are on or below the main diagonal.

A range $[u, v]$ contains another range $[x, y]$ if its corresponding point (v, u) is right and below (y, x). Hence, finding the most specific range containing a given destination address d corresponds to finding, for the point $p = (d, d)$ on the main diagonal, the topmost and leftmost point (v, u) that is right and below p. Note that if the most specific range exists, there is always a unique topmost-leftmost point.

Thus, solving the dynamic version of the IP lookup problem for prefix filters means to maintain a set of points in the plane for which we can carry out insertions and deletions of points and answer topmost-leftmost queries efficiently. Topmost-leftmost queries can be reduced to leftmost queries when ensuring that no two points have the same x-coordinate. This can be accomplished by mapping each point (x, y) to the point $(2^w x - y + 2^w - 1, y)$ [2]. Finding the leftmost point in a given south-grounded query range can be accomplished by answering a $MinXinRectangle(x_{left}, x_{right}, y_{top})$ query, which, for the rectangle bounded by x_{left} to the left, x_{right} to the right, y_{top} to the top and the x-axis to the bottom, returns the leftmost point from the given set (which corresponds to the most specific range filter).

When considering the more general case, i.e., where filters are specified by arbitrary ranges and not by prefixes, the most specific range for an incoming packet may not be defined. In this case, the filter set can be conflicting.

Definition 2. *A set of ranges R is **conflict-free** iff for each point p there is a unique range $r \in R$ such that $p \in r$ and for all ranges $s \in R$ that contain p, $r \subseteq s$.*

Definition 3. *Two ranges $r, s \in R$ are **in conflict** with respect to R if r and s partially overlap and there is a point p such that $p \in r \cap s$, but there is no range $t \in R$ such that $p \in t$ and $t \subset r$ and $t \subset s$.*

Definition 4. *Let $r, s \in R$ be two conflicting ranges.*

*We call the overlapping range $r \cap s$ the **resolve filter** for r and s with respect to R.*

Definition 5. *A set R of ranges is called **nonintersecting** if for any two ranges $r, s \in R$ either $r \cap s = \emptyset$ or $r \subseteq s$ or $s \subseteq r$.*

In other words, R is nonintersecting if any two ranges are either disjoint or one is completely contained in the other. It is obvious that a set of nonintersecting ranges is always conflict-free, i.e. for each packet (query point p) the most specific range in R containing p is well defined.

For a given set R of n one-dimensional nonintersecting ranges, the deletion of a range maintains the conflict-free property of R. The insertion of a new range $[u, v]$ however may create a conflict. In order to determine if a new range $r = [u, v]$ partially overlaps with any of the ranges in R, two conditions have to be verified:

1. $\exists s = [x, y] \in R : x < u \leq y < v$ (s left-overlaps r)
2. $\exists s = [x, y] \in R : u < x \leq v < y$ (s right-overlaps r)

In order to keep R nonintersecting, we have to refuse the insertion of such ranges that partially overlap with any of the ranges in R.

In this paper we present new data structures for the IP lookup and conflict detection problem. After describing related work, we present our solutions for online conflict detection in one dimension. In section 4 we investigate the hypothesis that a relaxed balancing scheme is better suited for search-tree based dynamic IP router tables than a scheme that utilizes strict balancing. In section 5 we present a sweepline algorithm for offline conflict detection in one dimension.

2 Related Work

A multitude of solutions for the representation of *static* as well as *dynamic* rule tables have been proposed. Typically in 'static' data structures, insertions and deletions are batched and the data structure is reconstructed from scratch as needed. Dynamic structures allow for efficient update operations.

Many of the proposed data structures are based on the binary trie structure, e.g. [3] [4] [5]. Lu and Sahni [6] propose a method to partition a static IP router table such that each partition is represented using a base structure such as a multibit trie [7] or a hybrid shape shifting trie [8].

Lee *et al.* [9] propose an algorithm which is based on the segment tree. Given n IP prefixes, their algorithm performs IP address lookup in $O(\log n)$ time. Their approach can also handle insertions and deletions of IP prefixes, but the segment tree has to be rebuilt from time to time in order to maintain lookup performance.

Lu and Sahni [10] develop an enhanced interval tree LMPBOB (longest matching prefix binary tree on binary tree) which permits lookup in $O(W)$ time, where W is the length of the longest prefix, and filter insertion and deletion in $O(\log n)$ time each.

Warkhede, Suri and Varghese [11] introduce an IP lookup scheme based on a multiway range tree with worst-case search and update time of $O(\log n)$, where n is the number of prefixes in the forwarding table.

Lu and Sahni [2] show that each of the three operations insert, delete and IP lookup may be performed in $O(\log n)$ time in the worst case using a data structure based on priority-search trees (see section 3).

The conflict detection problem occurs in two variants; the *offline* and *online* modes. Algorithms for the offline version are given a set of filters and report (and resolve) all pairs of conflicting filters. The online version, on the other hand, is a gradual build-up of a conflict-free set R; such that for every insertion and deletion of a range r, a check is made on the current status of R and if conflicts occur, the algorithms will offer solutions to resolve them and maintain the conflict-free property of R.

2.1 Online Conflict Detection and Resolution

If R is made up of arbitrary one-dimensional ranges, then both operations to insert and delete a range may lead to conflicts in the resulting set. In the case of an insertion, the new range may *left*- or *right*-overlap with one or more ranges in R, such that the overlapping range has no resolving subset. Similarly, a conflict may arise if we remove $t = r \cap s$ from R, because $t \in R$ is a range that resolves a conflict between $r, s \in R$. This online version of the problem is solved (in a rather complex way) by Lu and Sahni [2]. The core of their solution consists in a representation of a set of ranges and their endpoints, such that for any given query range $r = [u, v]$, one can efficiently decide whether or not there is a subset of ranges covering exactly this range (starting at u and ending at v). We turn to this issue in section 3.

Hari *et al.* [12] consider the 2-dimensional online version of conflict detection and resolution for sets of prefix filters. Each filter f is a pair $(f[1], f[2])$, where each field $f[i]$ is a prefix bit string. Assuming that the most-specific tie-breaker is applied, two filters f_1 and f_2 have a conflict iff

1. $f_2[1]$ is a prefix of $f_1[1]$ and $f_1[2]$ is a prefix of $f_2[2]$ or
2. $f_1[1]$ is a prefix of $f_2[1]$ and $f_2[2]$ is a prefix of $f_1[2]$

By adding a new filter which covers the region of intersection, the conflict between two filters can be eliminated. Hari *et al.* refer to this new filter as the resolve filter. The addition of the resolve filter maintains the conflict-free property of the filter set. For any packet that falls in the area of intersection, the resolve filter will be reported as the most specific filter that matches the packet.

2.2 Offline Conflict Detection and Resolution

Lu and Sahni [13] develop a plane-sweep algorithm to detect and report all pairs of conflicting 2D prefix filters under the most specific tie breaking rule. Their algorithm runs in time $O(n \log n + s)$, where n is the number of filters and s is the number of conflicts. Two filters $f, g \in F$ conflict iff an edge of f perfectly crosses an edge of g; that is, two edges *perfectly cross* iff they cross and their crossing point is not an endpoint. In other words, a proper-edge intersection between two ranges in F is a direct cause for conflict. This implies that all conflicts in F can be detected by computing all proper intersecting pairs of filters in F (and determining their regions of overlap). Therefore, we get an output-sensitive algorithm if all pairs of properly intersecting ranges in F are detected and reported. Lu and Sahni [13] further discuss the problem of resolving conflicts by adding a set of resolve filters to F. Such that for each conflicting prefix-pair $f, g \in F$, we add a new resolve filter $h = f \cap g$ to F. The resulting set of $F \cup resolve(F)$ is conflict-free.

3 Solutions Based on Priority Search Structures

Lu and Sahni propose a solution to the dynamic one-dimensional IP lookup problem (including online conflict detection) based on the well-known priority search tree (PST) [2]. They show that for prefix filters a single PST is sufficient to support insertion, deletion, and lookup in $O(\log n)$ time, where n is the number of filters. The lookup procedure depends on the *MinXinRectangle* query supported efficiently by PST, but does not require any other types of range queries.

Since prefix filters are always conflict-free, no conflict detection is needed. However, for filters specified as ranges the insertion requires a check for potential conflicts with existing filters within the set. (In case of a conflict, the insertion of the new range will be refused.) As long as the filters are restricted to nonintersecting ranges, Lu and Sahni solve the problem of detecting overlaps by employing an additional PST storing the same filters but using an orthogonal geometric mapping, thus keeping the overall insertion time in $O(\log n)$. However, the actual time for insertion and deletion, as well as the space requirements of the overall approach, are increased by roughly a factor of two.

In [14] we show that for nonintersecting ranges the second PST is not required, as all potential overlaps can be detected by two *MinXinRectangle* queries on the original structure. This improvement reduces the actual cost of insertion and deletion by half while leaving the search costs untouched. Moreover, we observe that Lu and Sahni's approach does not depend on PST but can employ any data structure that efficiently supports insertion, deletion, and *MinXinRectangle* queries on sets of points in the plane. Conceptually simpler data structures with the same asymptotic bounds include priority search pennants [15] and min-augmented range trees (see next section), both of which require less memory and are easier to keep balanced than the PST. Our experiments reported in [14] found that these structures are also faster for all three operations in actual implementations.

For general conflict-free ranges (this means that ranges may overlap as long as the intersections are covered by resolve sets), the approach in [2] requires a third support structure, in this case a collection of balanced binary search trees (Lu and Sahni use red-black trees) in order to keep track of so-called cover sets.

We were able to show that the improvement for nonintersecting ranges (i.e. removing the second structure) can be generalized to arbitrary conflict-free ranges, as long as only insertions are carried out and no deletions occur. The argument is analogous to that in [14], but requires the third support structure of [2] in order to check for the existence of cover sets (i.e. resolve filters) if the range to be inserted partially overlaps with any existing ranges.

However, if ranges are to be removed from the set as well, a secondary priority search structure might actually be indispensable in order to support those deletions in $O(\log n)$ asymptotic time. The reason is that it is harder to detect if a set of ranges is still conflict-free after removing a range than to find out whether inserting one new range in a set of conflict-free ranges will create a conflict.

4 Min-Augmented Range Trees

A min-augmented range tree (MART) [14] storing a set of points with pairwise different x-coordinates stores the points at the leaves such that it is a leaf-search tree for the x-coordinates of points. The internal nodes have two fields, a router field guiding the search to the leaves and a min-field. In the router field we store the maximum x-coordinate of the left subtree, and in the min-field we store the minimum y-coordinate of any point stored in the leaves of the subtrees of the node. Figure 1 shows a MART storing 16 points.

First we show how to answer a leftmost-, or *MinXinRectangle* $(x_{left}, \infty, y_{top})$ query [14]. That is, to find the point p with minimal x-coordinate in the semi-infinite range $x \geq x_{left}$ and with y-coordinate below the threshold value y_{top}. In order to find this point, we first carry out a search for the boundary value x_{left}. It ends at a leaf storing a point with minimal x-coordinate larger than or equal to x_{left}. If this point has a y-coordinate below the threshold value y_{top}, we are done. Otherwise we retrace the search path for x_{left} bottom-up and inspect the roots of subtrees falling completely into the semi-infinite x-range. These roots appear as right children of nodes on the search path. Among them we determine the first one from below (which is also the leftmost one and) which has a min-field value below the threshold y_{top}. This subtree must contain the answer to the *MinXinRectangle* $(x_{left}, \infty, y_{top})$ query stored at its leaf. In order to find it, we recursively proceed to the left child of the current node, if its min-field shows that the subtree contains a legal point, i.e. if its min-field is (still) below the threshold, and we proceed to the right child only, if we cannot go to the left child. Figure 1 shows the search path of the query *MinXinRectangle (35,80,34)*. We can find the most specific range in time which is proportional to the height of the underlying leaf-search tree. Hence it is desirable to maintain the underlying tree balanced. Rotations and the process of maintaining the augmented min-information are strictly local, hence we can freely choose an underlying balancing scheme for min-augmented range trees.

In order to accelerate lookup and update operations, routing tables must be implemented in a way that they can be queried and modified concurrently by several processes. If implemented in a concurrent environment there must be a way to prevent simultaneous reading and writing of the same parts of the data structure. A common strategy is to lock the critical parts. In order to allow a high degree of concurrency, only a small part of the tree should be locked at a time. Relaxed balancing has become a commonly used concept in the design of concurrent search tree algorithms. In relaxed balanced data structures, rebalancing is

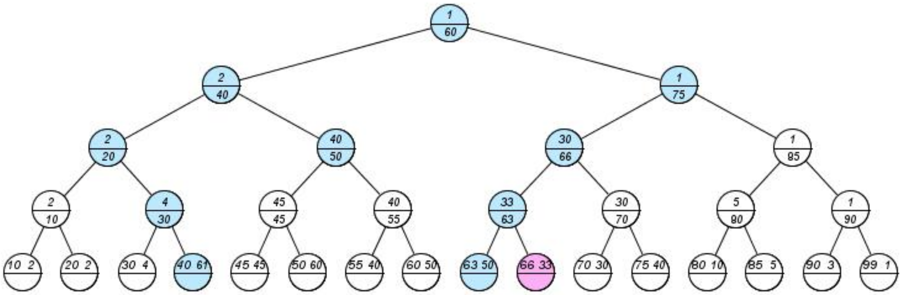


Fig. 1. The search path of the query *MinXinRectangle (35,80,34)* in a MART. Visited nodes are highlighted; the node with key *(66 33)* is the result returned by the query.

uncoupled from updates and may be arbitrarily delayed. This contrasts with strict balancing, where rebalancing is performed immediately after an update. Hanke [16] presents an experimental comparison of the strictly balanced red-black tree and three relaxed balancing algorithms for red-black trees, using the simulation of a multi-processor machine. The results indicate that the relaxed schemes have significant better performance than the strictly balanced version.

The locality property of the MART concerning the rebalancing of the tree enables us to decouple the update and rebalancing operations as will be shown in Section 4.1.

Motivated by Hanke's results, we proposed the relaxed min-augmented range tree and performed an experimental comparison of the strictly balanced MART and the relaxed balanced MART using real IPv4 routing data [17]. Some details are presented in the following section.

4.1 Relaxed Min-Augmented Range Trees

Instead of requiring that the balance condition is restored immediately after each update, the balance conditions are relaxed such that the rebalancing operations can be delayed and interleaved with search and update operations. Several relaxed balancing schemes have been proposed, especially for red-black trees [18] [19]. To relax the min-augmented range tree we utilized the scheme proposed in Hanke, Ottmann and Soisalon-Soininen [19]. This scheme can be applied to any class of balanced trees. The main idea is to use the same rebalancing operations as for the standard version of the tree. Red nodes may have *up-in* requests (resulting from a red-red conflict), black nodes may have *up-out* requests, and leaves may have removal requests. In contrast to the standard red-black tree, a red-red conflict is not resolved immediately after the actual insertion. The insertion process only marks a violation of the red constraint by an *up-in* request. The handling of a removal request may remove a black node from a path. This is marked by an *up-out* request. The task of the rebalancing processes is to handle all *up-in*, removal and *up-out* requests. For this the rebalancing operations of the standard red-black tree are used, which are split into small restructuring steps that can be carried out independently and concurrently at different locations in the tree.

The following subsection presents the experimental results of benchmarking the strict balanced MART with the relaxed balanced MART (RMART).

Experimental Results. To conduct the experiments reported in [17], we used real and up-to-date routing data that is supplied by the Advanced Network Technology Center at the University of Oregon within the framework of the Route Views Project[1]. Both the strict and relaxed balanced min-augmented range tree can be queried and modified concurrently by several processes. The difference is that updates must be performed serially in the strictly balanced MART, since the rebalancing is performed immediately after an update. Yet, an update can be performed concurrently with search operations. In the relaxed

[1] http://archive.routeviews.org/bgpdata/

balanced MART, several updates can be performed concurrently. Further, updates can be performed concurrently with search as well as with restructuring operations. Hence, it is expected that the relaxed version shows better behavior in scenarios with many updates. In the experiments, various scenarios with varying update/lookup ratios were simulated. If solely search operations were performed, the total time of the MART has shown to be on a par with the total time of the RMART. If also insert and delete operations were performed, then the higher the update/lookup ratio, and the higher the number of processes, the clearer did the RMART outperform the standard MART. The results showed that the speed-up per search operation grows with the update/lookup ratio. This confirmed the hypothesis that in the RMART, lookup queries are not as much delayed as in the standard version, since the rebalancing operations are postponed.

The faster a lookup query is answered, the smaller the chance that packets must be dropped due to buffer overload. Thus another advantage is that the rate of packet loss might be reduced.

The simulations were executed on a Sun Fire T2000 Server 6 Core 1.0 GHz UltraSPARC T1Processor, which has a Unified Memory Architecture (UMA), i.e., memory is shared among all cores. The higher the number of executing threads, in line with the available processors, the higher the speed-up. It would be interesting to investigate how the speed-up increases when the benchmark is performed on a multi-core processing maschine with more than six cores.

5 One-Dimensional Conflict Detection

As we have seen in section 3, Lu and Sahni [2] and our refinements in [14] show that online conflict detection can be achieved in $O(\log n)$ time, where n is the number of filters. In the following we present the idea of our solution for an offline conflict detection and resolution algorithm for static one-dimensional range tables [20]. The algorithm is based on the sweepline technique. A set of n ranges $R = \{r_1, \ldots, r_n\}$ defines at most $2n-1$ consecutive slabs defined by the endpoints of the ranges. Let $ep_0, ep_1, \ldots, ep_k$ be the boundaries of these slabs. We denote by $S_i \subseteq R$ the set of ranges which contain ep_i.

Definition 6. *Slab σ_i is conflict-free iff there is a shortest range $r \in S_i$ that contains ep_i, such that r is contained in all other ranges $S_i \subseteq R$ that contain ep_i.*

Let σ_i be a *non*-conflict-free slab. Then σ_i requires only a single “slab-resolve” filter $h_{\sigma_i} = [ep_i, ep_{i+1})$ to make it conflict-free.

The trivial solution to make a set conflict-free would be to add a “slab-resolve” filter for each of the slabs. Yet, this solution could add unnecessary filters since not every slab may require a resolve filter, e.g., when the set is already conflict-free. Hence the goal is to add only as many resolve filters as are needed to make the set conflict-free.

To detect and resolve all conflicts in the set, our proposed algorithm performs a single left to right sweep and at each endpoint it

- **(i)** determines the smallest filter r from all S_i stabbed by ep_i, and then
- **(ii)** checks if r is contained in all S_i to deduce that slab σ_i is conflict-free.
- **(iii)** If slab σ_i is non-conflict-free, then the resolve filter $h_{\sigma_i} = [ep_i, ep_{i+1})$ is added to the set.

The proposed algorithm achieves a worst case time complexity of $O(n \log n)$ and uses space $O(n)$. Further, when making use of partial persistence, this scheme also supports IP lookup. Ordinary data structures are *ephemeral* in the sense that an update on the structure destroys the old version, leaving only the new version available for use. A data structure is called *partially persistent* if all intermediate versions can be accessed, but only the newest version can be modified, and *fully persistent* if every version can be both accessed and modified. Refer to Driscoll *et al.* [21] for a systematic study of persistence.

In our sweepline approach, think of the x-axis as timeline. Note that the sets of line segments intersecting contiguous slabs are similar. As the boundary from one slab to the next is crossed, certain segments are deleted from the set and other segments are inserted. Over the entire time line, there are $2n$ insertions and deletions, one insertion and one deletion per segment. The idea is to maintain a data structure during Slab-Detect's sweep that stores for each slab the segments that cover the slab and also the resolve filter if the slab was found to be non-conflict-free. The first version of the structure refers to slab ep_0 and the last version refers to slab ep_k. Utilizing the path-merging technique [22] we are able to solve the conflict detection problem in time of $O(n \log n)$ while building the partially persistent structure, and then utilize it to answer IP lookup queries in $O(\log n)$ time.

6 Conclusions

In this paper we presented a survey of new data structures for the IP lookup and conflict detection problems. We presented the min-augmented range tree with a relaxed balancing scheme, allowing queries and updates to be carried out concurrently, while avoiding the costly delays of fixed balancing strategies. This structure can be used instead of the PST in the approach described in [2], which also comprised online conflict detection. Furthermore, we outlined additional improvements to this approach, saving space and update costs. For offline conflict detection and resolution we have described an efficient sweepline algorithm.

References

1. McCreight, E.M.: Priority search trees. SIAM J. Comput. 14(2), 257–276 (1985)
2. Lu, H., Sahni, S.: $O(\log n)$ dynamic router-tables for prefixes and ranges. IEEE Transanctions on Computers 53(10), 1217–1230 (2004)
3. Kim, K.S., Sahni, S.: Efficient construction of pipelined multibit-trie router-tables. IEEE Transactions on Computers 56(1), 32–43 (2007)

4. Song, H., Turner, J., Lockwood, J.: Shape shifting tries for faster IP route lookup. In: ICNP 2005: Proceedings of the 13th IEEE International Conference on Network Protocols (ICNP 2005), Washington, DC, USA, pp. 358–367. IEEE Computer Society, Los Alamitos (2005)
5. Ioannidis, I., Grama, A., Atallah, M.: Adaptive data structures for IP lookups. J. Exp. Algorithmics 10, Article No. 1.1 (2005)
6. Lu, W., Sahni, S.: Recursively partitioned static IP router-tables. In: 12th IEEE Symposium on Computers and Communications, pp. 437–442 (2007)
7. Srinivasan, V., Varghese, G.: Faster IP lookups using controlled prefix expansion. In: SIGMETRICS 1998/PERFORMANCE 1998: Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, pp. 1–10. ACM Press, New York (1998)
8. Lu, W., Sahni, S.: Succinct representation of static packet classifiers. In: 12th IEEE Symposium on Computers and Communications, pp. 1119–1124 (2007)
9. Lee, I., Park, K., Choi, Y., Chung, S.K.: A simple and scalable algorithm for the IP address lookup problem. Fundamenta Informaticae 56(1,2), 181–190 (2003)
10. Lu, H., Sahni, S.: Enhanced interval trees for dynamic IP router-tables. IEEE Transactions on Computers 53(12), 1615–1628 (2004)
11. Warkhede, P., Suri, S., Varghese, G.: Multiway range trees: scalable IP lookup with fast updates. Computer Networks 44(3), 289–303 (2004)
12. Hari, A., Suri, S., Parulkar, G.: Detecting and resolving packet filter conflicts. In: INFOCOM 2000: Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 1203–1212. IEEE Press, Los Alamitos (2000)
13. Lu, H., Sahni, S.: Conflict detection and resolution in two-dimensional prefix router tables. IEEE/ACM Transactions on Networking 13(6), 1353–1363 (2005)
14. Lauer, T., Ottmann, T., Datta, A.: Update-efficient data structures for dynamic IP router tables. International Journal of Foundations of Computer Science 18(1), 139–161 (2007)
15. Hinze, R.: A simple implementation technique for priority search queues. In: International Conference on Functional Programming, pp. 110–121 (2001)
16. Hanke, S.: The performance of concurrent red-black tree algorithms. In: Vitter, J.S., Zaroliagis, C.D. (eds.) WAE 1999. LNCS, vol. 1668, pp. 286–300. Springer, Heidelberg (1999)
17. Maindorfer, C., Bär, B., Ottmann, T.: Relaxed min-augmented range trees for the representation of dynamic IP router tables. In: 13th IEEE Symposium on Computers and Communications, pp. 920–927 (2008)
18. Ottmann, T., Soisalon-Soininen, E.: Relaxed balancing made simple. Technical Report 71, Institut für Informatik, Albert-Ludwigs-Universität Freiburg (1995)
19. Hanke, S., Ottmann, T., Soisalon-Soininen, E.: Relaxed balanced red-black trees. In: Bongiovanni, G., Bovet, D.P., Di Battista, G. (eds.) CIAC 1997. LNCS, vol. 1203, pp. 193–204. Springer, Heidelberg (1997)
20. Maindorfer, C., Mohamed, K.A., Ottmann, T., Datta, A.: A new output-sensitive algorithm to detect and resolve conflicts in Internet router tables. In: INFOCOM 2007. 26th IEEE Conference on Computer Communications, pp. 2431–2435 (2007)
21. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making data structures persistent. In: STOC 1986: Proceedings of the eighteenth annual ACM symposium on Theory of computing, pp. 109–121. ACM Press, New York (1986)
22. Mohamed, K.A., Langner, T., Ottmann, T.: Versioning tree structures by path-merging. In: Preparata, F.P., Wu, X., Yin, J. (eds.) FAW 2008. LNCS, vol. 5059, pp. 101–112. Springer, Heidelberg (2008)

Group-Level Analysis and Visualization of Social Networks

Michael Baur[1], Ulrik Brandes[2], Jürgen Lerner[2], and Dorothea Wagner[1]

[1] Faculty of Informatics, Universität Karlsruhe (TH), KIT
[2] Department of Computer & Information Science, University of Konstanz

Abstract. Social network analysis investigates the structure of relations amongst social actors. A general approach to detect patterns of interaction and to filter out irregularities is to classify actors into groups and to analyze the relational structure between and within the various classes. The first part of this paper presents methods to define and compute structural network positions, i. e., classes of actors dependent on the network structure. In the second part we present techniques to visualize a network together with a given assignment of actors into groups, where specific emphasis is given to the simultaneous visualization of micro and macro structure.

1 Network Analysis

Social network analysis (SNA) [54] is an established, active, and popular research area with applications in sociology, anthropology, organizational studies, and political science, to name a few. In a nutshell, SNA analyzes the structure of relations among (social, political, organizational) actors. While the type and interpretation of actors and relations—as well as the theoretical background of network analysis—varies from application to application, many network analysis *methods* are nevertheless applicable in rather general settings.

In order to abstract from the particular application context, we assume that networks are represented by *graphs* $G = (V, E)$, where V is a set of *vertices*, encoding the actors, and E is a set of *edges* (also called *ties* or *links*), encoding the relation among actors. Edges may be directed, undirected, or of mixed type. Furthermore, vertices and edges may have various attributes encoding, e. g., the type of actors or relations as well as the strength of relations.

Network analysis methods can be classified with respect to the level of granularity of the analyzed objects (compare [11]):

- **Element-level** methods analyze properties of individual vertices and edges, such as importance (*centrality*).
- **Group-level** analysis determines specific subsets of vertices. These methods include the computation of densely connected groups (*clustering*) and the computation of structural roles and positions (*blockmodeling* or *role assignment*, see Sect. 2).

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 330–358, 2009.

- **Network-level** analysis is interested in global properties of the network, such as density, degree-distributions, transitivity, or reciprocity; as well as in the development of random graph models that are plausible for empirical networks.

In this chapter we focus on group-level network analysis. For surveys encompassing all levels of network analysis, see, for instance, [54] and [11]. In the remainder of this section, we briefly introduce a software to analyze and visualize social networks and state common notation. Thereafter, in Sect. 2, we give an overview of state-of-the-art methods for role assignment and present our own contribution to this field. Section 3 details a visualization technique for networks on which a partition of the vertices is already given (e. g., from clustering, role assignments, or extrinsic vertex-attributes) and where the analyst wants to see the interplay between fine-grained (vertex-level) and coarse-grained (group-level) structures.

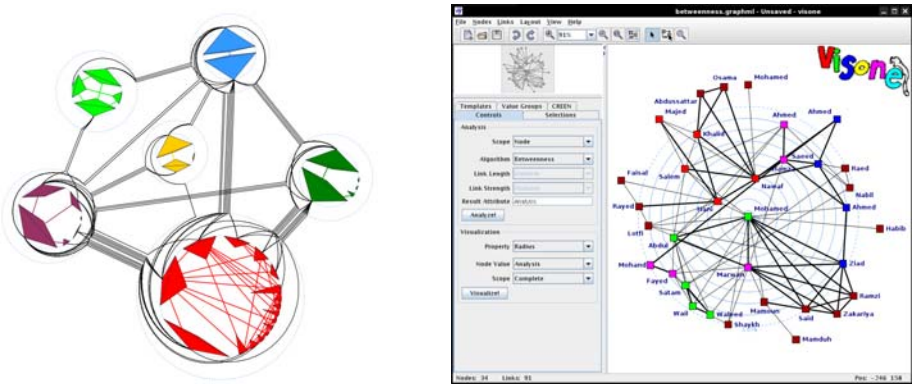
1.1 visone – Software for the Analysis and Visualization of Social Networks

Along with the increased relevance of network analysis and the growing size of considered networks, adequate software for social network analysis is becoming more and more important. As part of our project we provide the software tool visone[1], aiming to bring together efficient algorithms for methods of analysis and suitable graph drawing techniques for the visualization of networks. Besides our original work, we have included novel algorithms developed by other members of our groups at the universities of Karlsruhe and Konstanz in order to cover fields like centrality indices [10], clusterings [27], and spectral layouts [24]. The functionality is completed by well-known commonly-used methods.

visone is not only intended as a testbed for the work of our groups but also as an everyday tool for students and researchers in network analysis. Therefore, we adapt all algorithms to a consistent and comprehensive graph model and put in great efforts to provide a simple but flexible user interface hiding unnecessary complexity. In contrast to common tools which present to the user only a matrix representation of the data, we build on the expressive and explanatory power of graph layouts and provide a complete graphical view of the network (see Fig. 1(b)). Observations indicate that users enjoy the playful nature of our approach.

Visualizing social networks is more than simply creating intriguing pictures, it is about generating learning situations: *"Images of social networks have provided investigators with new insights about network structure and have helped them communicate those insights to others"* [25]. Additionally, inappropriate drawings of networks are misleading or at least confusing. Therefore, we pay special attention to the visualization of the networks. Selected general graph layout algorithms provide an uncluttered view on the network and reveal its overall

[1] visone is available free of charge for academic and non-commercial purpose from the homepage `http://visone.info`



Analysis

Scope: Node

Algorithm: Betweenness

Link Length

Link Strength

Result Attribute: Analysis

Analyze!

Visualization

Property: Radius

Node Value: Analysis

Scope: Complete

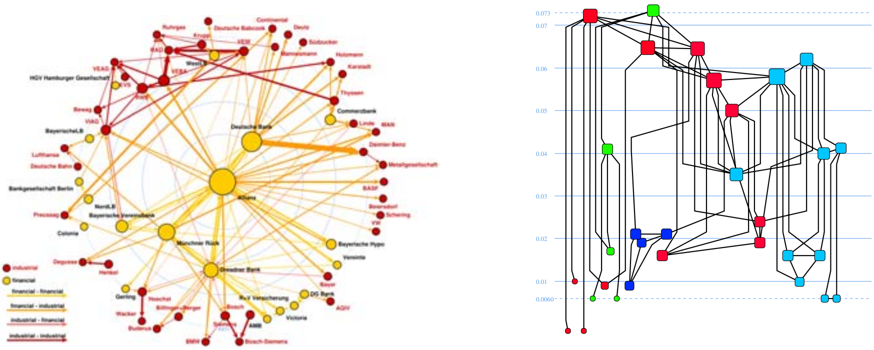
Visualize!

Nodes: 34 Links: 91

(a) multi-circular visualization

(b) main window of visone

Fig. 1. (a) Multi-circular visualization of a network consisting of six groups. The group structure is clearly visible. Additionally, the height and width of the vertices reflects the number of connections within and between groups. (b) The most notable features of the main window of visone are the large and detailed view of the graph, the small overview, and the control pane on the left hand.



(a) radial visualization

(b) status visualization

Fig. 2. Examples of the radial and the status visualization. The positions of the vertices depict centrality measures. Additional information is reflected by the color, shape, size, and width of the vertices and edges.

structure, but the unique feature of visone are the analytic visualizations which exactly depict analysis results, like centrality scores and clusterings, by means of tailored and suggestive graph layouts (see Figs. 1(a) and 2). Combinatorial models of these visualizations allow for the optimization of esthetic properties to improve the expressiveness and exploratory power without changing their analytic signification.

1.2 Basic Notation

Let $G = (V, E)$ be a directed or undirected graph with $n = |V|$ vertices and $m = |E|$ edges. A *partition* of G is a subdivision of the vertex set V into pairwise disjoint, non-empty subsets $V = V_1 \dot{\cup} \ldots \dot{\cup} V_k$. In addition to this explicit definition, a partition can be given by an equivalence relation on V or by a surjective mapping $\rho \colon V \to \{1, \ldots, k\}$ of vertices to vertex-classes (called *partition assignment*). These three definitions are mutually in a canonical one-to-one correspondence up to permutation (re-labeling) of classes, see [39], and we typically identify the class i with the set V_i.

A partition assignment $\rho \colon V \to \{1, \ldots, k\}$ defines a smaller graph $Q(G, \rho) = (\mathcal{V}, \mathcal{E})$, called the *quotient graph*, encoding which classes are connected, by setting

$$\mathcal{V} = \{1, \ldots, k\} \text{ and } \mathcal{E} = \{(\rho(u), \rho(v))\,;\ (u, v) \in E\}\ . \tag{1}$$

2 Structural Positions in Networks

The notion of *(structural) position* is fundamental in social network analysis, see for example [54,9]. Actors are said to occupy the same position if they have identical patterns of ties to other actors and the task of determining such classes of actors is referred to as *blockmodeling* or *role assignment*. For instance, by this definition university professors would occupy the same structural position if they have identical patterns of ties to students, secretaries, other professors and so on. Note that this definition of position dependent on the network structure contrasts to more traditional notions of social positions, such as defining the position of professors dependent on the type of contract that they have with their university. In this paper the term *position* always refers to structural position. Various types of role assignment differ in how they operationalize the notion of *identical patterns of ties to other actors* and how they account for deviation from perfectly identical patterns. We continue by reviewing established previous notions for role assignment and outline where the newly proposed *structural similarities* fit in.

2.1 Previous Work on Role Assignments

Basic Notation. A *role assignment* $r \colon V \to \{1, \ldots, k\}$ of a (directed or undirected) graph $G = (V, E)$ is given by a partition of its vertex set V. In context of role assignments, vertex-classes are also referred to as *positions* and the quotient graph is called *role graph*. A role assignment r defines k^2 submatrices, called *blocks*, of G's adjacency matrix A. The block associated to class C and D, denoted by $A[C, D]$, is the $|C| \times |D|$ submatrix of A whose rows correspond to the vertices in C and whose columns correspond to the vertices in D.

If a role graph (i. e., a hypothesis for the role structure of a network) is given, the problem of determining a role assignment that yields this role graph is called a *role assignment problem*, or *prespecified blockmodeling*.

Discrete Approaches. Specific types of role assignments are obtained by requiring that vertex partitions must satisfy specific compatibility constraints with respect to the graph structure. An important distinction between various constraints is whether they require equivalent vertices to be connected to the *same* others —illustrated in Fig. 3 (*left*)—or just to *equivalent* (but not necessarily the same) others—illustrated in Fig. 3 (*right*).

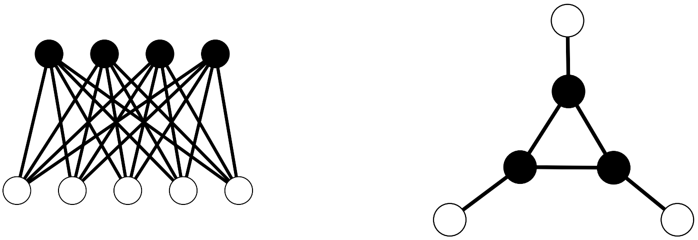


Fig. 3. Two graphs with vertex partitions indicated by the coloring. *Left*: Equivalent vertices have identical neighborhoods. *Right*: Equivalent vertices have equivalent but non-identical neighborhoods.

Neighborhood Identity. The most basic approach defines vertices as *structurally equivalent* [41] if they have identical neighborhoods, i. e., if they are connected to exactly the same others; compare Fig. 3 (*left*). An equivalence is structural if and only if all induced blocks are either complete (consisting only of ones) or zero.

Structural equivalence (SE), however, is too strict and does not well match intuitive notions of network position. Coming back to the example from the beginning of Sect. 2, SE would assign the same position to professors only if they are connected to the *same* students, secretaries, and other professors. Work discussing the insufficiency of SE includes [48], [9], and [42].

Neighborhood Equivalence. To capture more general situations, SE has been relaxed by requiring that vertices occupying the same position must only be connected to the same positions—independent on whether these positions are occupied by the same vertices or different vertices. Thus, two professors would be assigned to the same position if they both have the same patterns of relations to *some* (but not necessarily the same) students, secretaries, and other professors; compare Fig. 3 (*right*) where the black vertices are all connected to some (but different) white vertices. Mathematical formalizations of this idea include *regular equivalence*, *automorphic equivalence*, and *exact regular equivalence.*

A partition is called *regular* [55,22] if for every two of its classes C and D it holds that, whenever one vertex in C has a neighbor in D, then every vertex in C has a neighbor in D. Equivalently, a partition is regular if every induced block is either empty or it has at least one non-zero entry in every row and in every column. A partition is called *exact regular* [22] or *equitable* [31] if for every two of its classes C and D, all vertices in C have the same number of neighbors in D. Equivalently, a partition is exact regular if for every block B, there are two numbers r_B and c_B such that all rows of B sum up to r_B and all columns of B sum up to c_B. Two vertices u and v are called *automorphically* equivalent if there is a graph automorphism mapping u to v. The notion of automorphic equivalence is quite established in algebraic graph theory (e. g., [31]); work using this concept in social network analysis includes [9]. A structural partition is automorphic, an automorphic partition is equitable, and an equitable partition is regular.

Applicability for Social Network Analysis. The requirement for equitable partitions (and thus for automorphic and structural equivalence) is too strong for social networks (and other irregular, empirical data); due to deviations from ideal structural models, the resulting partitions will have singletons or very small classes. On the other hand, the maximal regular equivalence is often trivial as well; on undirected graphs it corresponds to the division into isolates and non-isolates. Determining non-trivial regular equivalences with a prespecified number of equivalence classes is NP-hard [23]. Regular, equitable, and automorphic equivalence is not robust against the addition or deletion of single edges (e. g., caused by noise or measurement errors); destroying the equivalence of one pair of vertices by adding/deleting an edge can have a cascading effect destroying equivalence of some of their neighbors, second-order neighbors, and so on. In conclusion, structural, automorphic, equitable, and regular partitions have limited applicability for the analysis of empirical data.

Real-valued Degrees of Similarity. To overcome (some of) the abovementioned problems, a formalization of role assignment should not only define *ideal* types of equivalence (such as regular, automorphic, or structural) but also clarify how to measure deviation from ideality (cf. [54]). Seen from a different angle, a formalization of role assignment should not only provide the decision between equivalent and non-equivalent but rather it should yield a *degree of similarity* of vertices.

Relaxing Structural Equivalence to Neighborhood Overlap. Defining degrees of structural equivalence is straightforward, although various different possibilities to do so exist. In most cases similarity is defined by measuring the overlap of the neighborhoods of two vertices and normalizing this measure in an appropriate way. Examples include taking the number of vertices in the intersection of neighborhoods divided by the number of vertices in the union of neighborhoods, the cosine of the angle between the two neighborhood vectors, and the correlation between two neighborhood vectors; see [54] for a more detailed discussion.

The so-defined measures of vertex similarity yield stable and efficient methods for the analysis of social networks. However, they do not overcome the inherent insufficiency of structural equivalence discussed earlier and mentioned in [48], [9], and [42]. In our running example, two professors would only be recognized as similar if they are both in relation to many common others (students, secretaries, and other professors); in contrast, two professors that mostly interact with disjoint alters would not be assigned similar network positions—even if their patterns of relations are similar.

Relaxing Neighborhood Equivalence. To combine the generality of notions of neighborhood equivalence (e. g., regular, equitable or automorphic partitions) with the robustness and empirical applicability of similarity measures (as opposed to equivalence), there is a need for relaxing neighborhood equivalence. However, previously proposals to do so are unsatisfactory for different reasons. In the following we briefly sketch one proposal for relaxing regular equivalence before we turn to an extended discussion of the newly proposed structural similarities.

Batagelj *et al.* [7] proposed an optimization algorithm to determine, for a fixed number k, a k-partition that has the least number of deviations from regularity; their method belongs to the framework of *generalized blockmodeling* [19]. Recall that a partition is regular if and only if every induced block is either zero or has at least one non-zero entry in each row and in each column. Measuring deviation from regularity of a specific partition is done by counting for each induced block the number of ones on one hand and the number of all-zero rows plus the number of all-zero columns on the other hand. The smaller of these two numbers is considered as the deviation from regularity of this particular block and by summing over all blocks one obtains the deviation from regularity of the partition. The associated optimization problem consists in finding the k-partition with the least deviation from regularity for a given graph. Since it is NP-complete to decide whether a graph admits a regular equivalence relation with exactly k classes [23], the abovementioned optimization problem is NP-hard as well; [7] proposed a local optimization algorithm to compute heuristically a k-partition with a small error. However, this approach is unsatisfactory for its computational inefficiency and lack of understanding of when the algorithm converges to a global optimum. In Sect. 2.2 we propose an alternative relaxation of neighborhood equivalence that enjoys more desirable properties.

2.2 Structural Similarity

The blockmodeling approach from [7] relaxed the *constraint* on partitions from being regular to having the least deviations from regularity. Structural similarities, in contrast, are obtained from equitable partitions (exact regular equivalence) by relaxing the *partitions* and keeping the constraint.

Basic Definitions. A discrete partition of n vertices in k can be represented by its characteristic matrix $P \in \mathbb{R}^{k \times n}$, where the entry $P_{iv} = 1$ if vertex v is in class i and zero else. Thus the degree of membership of a specific vertex to a specific class is either zero or one. Relaxations of partitions are obtained by allowing real-valued degrees of membership.

Definition 1 ([13]). *Given a graph on n vertices, a matrix* $P \in \mathbb{R}^{k \times n}$ *is called a* projection *(of dimension k) if* $PP^{\mathrm{T}} = \mathrm{id}_k$. *The entry* $P_{iv} = 1$ *is called the* degree of membership *of vertex v in class i; a row of P is considered as a* real-valued class *of vertices in which all n vertices have varying degrees of membership. The real $n \times n$ matrix $S = P^{\mathrm{T}}P$ is called the associated* similarity*. The entry S_{uv} is called the* similarity *of vertices u and v.*

The uv'th entry of $S = P^{\mathrm{T}}P$ is the inner-product of the two k-dimensional membership vectors of u and v, respectively. This value is large if u and v are to a high degree in the same classes. The constraint $PP^{\mathrm{T}} = \mathrm{id}_k$ on projections ensures that classes are orthogonal (independent) and normalized. Projections and similarities are in a canonical one-to-one correspondence, up to orthogonal transformations of the rows of the projection [13].

Just as a vertex partition defines a smaller graph encoding the adjacency of vertex classes—compare Eq. (1)—a similarity on a graph induces a *quotient* encoding the (weighted) adjacency of (real-valued) classes.

Definition 2 ([13]). *Let G be a graph with adjacency matrix $A \in \mathbb{R}^{n\times n}$ and $P \in \mathbb{R}^{k\times n}$ a projection. Then, G and P induce a $k \times k$ matrix B by setting $B = PAP^{\mathrm{T}}$. The (weighted) graph G/P on k vertices that is determined by its adjacency matrix B is called the* quotient *of G modulo P.*

Just as equitable partitions are partitions satisfying a certain compatibility constraint with the network structure, structural similarities are similarities satisfying a structural constraint.

Definition 3 ([13]). *A similarity S and its associated projection are called* structural *for a given graph with adjacency matrix A if $SA = AS$.*

The compatibility constraint $SA = AS$ can be used as an alternative definition of equitable partitions, see [40]. Indeed, if a similarity S is induced by a discrete partition $\mathcal{P}$, then S is structural if and only if $\mathcal{P}$ is equitable [13]. Thus structural similarities do neither relax nor modify the constraint of equitable partitions; they rather generalize discrete partitions to the larger class of similarities.

Characterization and Computation. The key to derive several desirable properties of structural similarities is the following characterization theorem that links structural similarities of a graph G to spectral properties of G's adjacency matrix. General references introducing the use of linear algebra methods in graph theory include [18,31].

Theorem 1 ([13]). *A similarity S is structural for an undirected graph with adjacency matrix A if and only if the image (i. e., the column-space) of S is spanned by eigenvectors of A.*

For directed graphs one has to distinguish between similarities that are structural with respect to outgoing edges, incoming edges, or both. Theorem 1 then holds if "spanned by eigenvectors" is replaced by "invariant subspace" and, depending on the type of structurality, "column-space" by "row-space" or "column-space and row-space;" see [40] for details.

Theorem 1 reduces the problem of computing structural similarities to that of computing eigenvectors (or invariant subspaces in the directed case). Many efficient numerical algorithms exist for these problems [32].

2.3 Structural Similarities Compared to Traditional Spectral Techniques

Orthogonal projections to low-dimensional subspaces that are spanned by eigenvectors are a frequent tool in many data analysis and graph partitioning applications. Concrete examples include latent semantic indexing [46], Web search [1], collaborative filtering [4], learning mixtures of distributions [52], analysis of the autonomous systems graph [30], graph clustering [35], random graph coloring [2], spectral graph partitioning [45], and graph bisection [16].

Typically, these methods project onto the eigenvectors corresponding to the (few) eigenvalues with the largest absolute values. (We will refer to these methods as *traditional spectral methods* in the following.) Thus, by Theorem 1, these methods compute special cases of structural similarities; the latter are not restricted to projecting to the largest eigenvalues but can choose all subsets.

We argue below that the difference between these two approaches is conceptually the same as between the requirements of identical vs. equivalent neighborhoods for equivalent vertices (compare Sect. 2.1). Thus, traditional spectral methods can be seen as relaxations of neighborhood *identity*, whereas structural similarities have been characterized as relaxations of equitable partition (exact regular equivalence) and, hence, of neighborhood *equivalence*.

An Illustrating Example. For instance, the (structural) partition shown in Fig. 3 (*left*) can be computed by projecting to the two eigenvalues ± 4.47, which have the maximal absolute values (the others are a seven-fold eigenvalue at 0). In contrast, the (equitable) partition shown in Fig. 3 (*right*) can be computed by projecting to the two eigenvalues 2.41 and -0.41, out of the set of eigenvalues

$$\mathbf{2.41}, 0.62, 0.62, -\mathbf{0.41}, -1.62, -1.62 \ .$$

Thus, restricting spectral algorithms to projections to the maximal eigenvalues yields methods that can not even identify some—intuitively outstanding—automorphic equivalences.

The General Case. Let A be the adjacency matrix of an undirected graph, S the matrix of a structural similarity for this graph, and u and v two vertices. The value $\|A(u-v)\| = \|A(u) - A(v)\|$ is a measure for the difference of the neighborhoods of u and v. Thus, u and v have *almost identical neighborhoods* if $\|A(u-v)\|$ is small. Similarly, u and v are considered as *almost equivalent* by S if $\|S(u-v)\|$ is small. We clarify below that traditional spectral methods optimize the property "$\|S(u-v)\|$ *is small if and only if* $\|A(u-v)\|$ *is small*", i. e.,

> u and v are considered as almost equivalent by S if and only if u and v have almost identical neighborhoods.

To make this precise, let $x_1, \dots, x_n$ be orthonormalized eigenvectors of A with associated eigenvalues $\lambda_1, \dots, \lambda_n$ which are ordered such that S projects to the first k eigenvectors. (Thus, if S is determined by traditional spectral methods the first k eigenvalues are those with maximal absolute values.) Further, let c_1 and c_2 be defined by

$$c_1 = \max_{i=1,\ldots,k} 1/|\lambda_i| \qquad \text{and} \qquad c_2 = \max_{i=k+1,\ldots,n} |\lambda_i| \ .$$

(Note that c_1 is defined only if S does not project to an eigenvalue $\lambda_i = 0$, which can be safely assumed for traditional spectral methods.) If k is given, then traditional spectral methods chose the structural projection of dimension k that minimizes c_1 and c_2 over all structural projections of dimension k. Let y be any vector of norm less than or equal to $\sqrt{2}$ and $y = \sum_{i=1}^{n} a_i x_i$ for uniquely determined real values a_i. It is

$$\|S(y)\|^2 = \sum_{i=1}^{k} a_i^2 \leq c_1^2 \sum_{i=1}^{k} (a_i\lambda_i)^2 \leq c_1^2 \|A(y)\|^2 \qquad \text{and} \tag{2}$$

$$\|A(y)\|^2 = \sum_{i=1}^{k} (a_i\lambda_i)^2 + \sum_{i=k+1}^{n} (a_i\lambda_i)^2 \leq \|A\|_2^2 \|S(y)\|^2 + 2c_2^2 \ . \tag{3}$$

By taking $y = u - v$ for the two vertices u and v, we obtain from (2) and (3) the following two properties for a structural similarity S.

1. Assume that S does not project to an eigenvalue $\lambda_i = 0$. If $\|A(u) - A(v)\|$ is small, then $\|S(u) - S(v)\|$ is small, i. e., vertices with almost identical neighborhoods are considered as almost equivalent by S. Furthermore, the ratio $\|S(u-v)\|/\|A(u-v)\|$ is bounded from above by c_1 which is minimized by traditional spectral methods.
2. Conversely, if $\|S(u) - S(v)\|$ is small then $\|A(u) - A(v)\|$ is bounded by $\sqrt{2}c_2$ plus a small $\varepsilon > 0$, i. e., if vertices are seen as almost equivalent by S, then their neighborhoods can differ by no more than $\sqrt{2}c_2$. Traditional spectral methods minimize c_2, i. e., those methods recognize only vertices with almost identical neighborhoods as almost equivalent.

It is important to note that these two properties can cause traditional spectral methods to miss some structure of the graph. Vertices may have a high structural similarity (e. g., they may be even automorphically equivalent) without having almost identical neighborhoods; compare Fig. 3.

2.4 The Role-Assignment Problem

For a given graph there is a huge set of structural similarities. Selecting the most appropriate one (or at least narrowing the choice) can be done by specifying how vertex classes (corresponding to structural positions in the network) are connected. Section 2.6 illustrates how the following theorem can be applied in the analysis of empirical data.

Theorem 2 ([13]). *Let G be an undirected graph with adjacency matrix $A \in \mathbb{R}^{n\times n}$ and R be a graph with adjacency matrix $B \in \mathbb{R}^{k\times k}$. Then, there is a structural projection $P \in \mathbb{R}^{k\times n}$ such that $B = PAP^{\mathrm{T}}$ if and only if B is symmetric* *and the characteristic polynomial of B divides the characteristic polynomial of A. In this case, the image of the similarity associated to P is generated by eigenvectors of A associated to the eigenvalues of B.*

In addition to its practical value, Theorem 2 also shows that the role assignment problem, which is computationally intractable for discrete notions of network position [23], is efficiently solvable for structural similarities.

2.5 Stability and Non-arbitrariness

A structural similarity S is associated to a set of eigenvalues of the graphs adjacency matrix A, namely the eigenvalues of $B = PAP^{\mathrm{T}}$. If all of these eigenvalues have the same multiplicity in B as they have in A, we call S a *simple* structural similarity. We show in this section that simple structural similarities enjoy two properties—that of being invariant under automorphisms and that of depending continuously from the adjacency matrix.

Non-arbitrariness. We say that a similarity (and hence more specifically a partition) is *non-arbitrary* if it is only derived from the graph's structure and not from a particular labeling of vertices. This, in turn, is formalized by being invariant under graph automorphisms, where *invariant under an automorphism* φ means that the similarity of every pair of vertices u and v is the same as the similarity of their images $\varphi(u)$ and $\varphi(v)$ (this is made precise in Def. 4). Figure 4 shows a small network together with an automorphism invariant partition (*left*) and a partition that is not automorphism invariant (*right*).

Definition 4. *Let $G = (V, E)$ be a graph. A similarity S is called* automorphism invariant *(for G) if for every two vertices $u, v \in V$ and every graph automorphism $\varphi \colon V \to V$ of G it is $S_{uv} = S_{\varphi(u)\varphi(v)}$.*

Theorem 3 ([40]). *A simple structural similarity is automorphism invariant.*

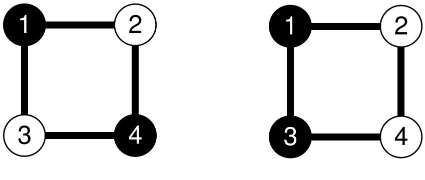


Fig. 4. Two different colorings on a graph with spectrum $\{2, 0, 0, -2\}$. *Left*: The coloring corresponds to the structural projection onto $\{2, -2\}$ and is automorphism invariant. This coloring reflects the unique bipartition of the graph and is therefore well justified by the graph structure. *Right*: The coloring corresponds to a structural projection onto $\{2, 0\}$ (only one eigenvector with eigenvalue 0 is taken). This coloring is not automorphism invariant (e. g., transposing 2 and 3 changes the partition). Intuitively, it seems to be arbitrary and not justifiable by the graph structure that Vertex 1 should be more similar to 3 than to 2, as suggested by the partition on the right.

The converse of Theorem 3 would hold if we took a weaker definition for automorphisms, see [40]. Theorem 3 also gives a criteria when equitable partitions are automorphism invariant since these are special cases of structural similarities.

Stability. A further desirable property of structural similarities is that their robustness to changes in the input data (e. g., caused by errors or dynamics) can be well-characterized. The following definition corresponds to the definition of the *separator*, known in matrix perturbation theory [51].

Definition 5. *Let S be a simple structural similarity for an undirected graph with adjacency matrix A. Let B be the induced quotient, Λ_B the spectrum of B, and Λ_A the spectrum of A. The positive real number*

$$\sigma(S) = \min\{|\lambda_1 - \lambda_2|\,;\ \lambda_1 \in \Lambda_B,\, \lambda_2 \in \Lambda_A \setminus \Lambda_B\}$$

is called the stability *of S.*

For a more general definition including the case of directed graphs see [51]. A large value $\sigma(S)$ guarantees resistance to perturbations of the input matrix A. Many error bounds can be given differing in the matrix norms that are used to measure the deviation and in the assumptions on the form of the error. See [51, Chapts. IV and V] for a representative set of error bounds. Examples of concrete error bounds for structural similarities under different assumptions are given in [12] and [14].

2.6 Applications of Structural Similarity

A structural similarity yields a low-dimensional embedding for the vertices of a graph. There are several ways for post-processing this embedding to obtain insights into the data. The first way is to apply a distance-based clustering procedure to the vertices in the low-dimensional embedding to obtain a discrete vertex partition. We followed this approach in [14], where it has been shown that the framework of structural similarities yields more general algorithms for random graph coloring. While traditional approaches can only deal with random graph models where edge probabilities are uniform, the newly proposed algorithm can handle models with non-uniform probabilities, provided that each vertex has the same expected number of neighbors from each class of differently colored vertices. This generalizations is conceptually the same as the relaxation from neighborhood identity to neighborhood equivalence; compare Sect. 2.1 and Sect. 2.3.

A second way to deal with the low-dimensional embedding is not to round it to a discrete partition but rather to apply multidimensional scaling (MDS) techniques to *visualize* the result in two or three dimensional space. Vertices that occupy (almost) the same positions will then be drawn close together and vertices that occupy very different positions will be far apart. The advantage of such a continuous representation of vertex positions is that we can accommodate with vertices that stand between two or more positions (that play more than one role).

We argue that such situations arise often in real-world data and forcing vertices to be members of one and only one class would then produce sub-optimal results. We will follow this approach to develop analysis and visualization methods for conflict networks.

Analysis and Visualization of Conflict Networks. The framework of structural similarities is especially convenient to develop methods for the analysis of large, noisy, empirical data sets. In [12] we presented a method to visualize dynamic networks of conflict between political actors. We review its essentials here, since this method is a good way to illustrate the use of Theorem 2.

Conflict networks are networks where the edges have a negative or hostile interpretation, such as criticism, accusations, or military engagements. Weighted edges arise from time-stamped events between the actors involved. Given a conflict network we generate a dynamic visualization that shows which group of actors is in opposition to which other group, which actors are most involved in conflict, and how do conflicts emerge, change their structure, and fade out over time. The example data set is from the Kansas Event Data System (KEDS) [49] and consists of approximately 78,000 dyadic events between political actors in the Balkans region.

We make the assumption that actors are loosely grouped together such that conflicts occur mostly between members of different groups. Thus, an actor is a member of one out of k classes to the extent that it has conflicts with members of the other classes.

We describe our method for the situation when there are only two groups that are mutually in conflict. To obtain a real-valued assignment of actors to the two groups we consider the quotient R_{cw} shown in Fig. 5 (*left*). The eigenvalues of R_{cw} are

$$\lambda = c + w \qquad \text{and} \qquad \mu = c - w \ .$$

From a different perspective the edge-weights of the quotient R_{cw} are determined by its two eigenvalues λ and μ as

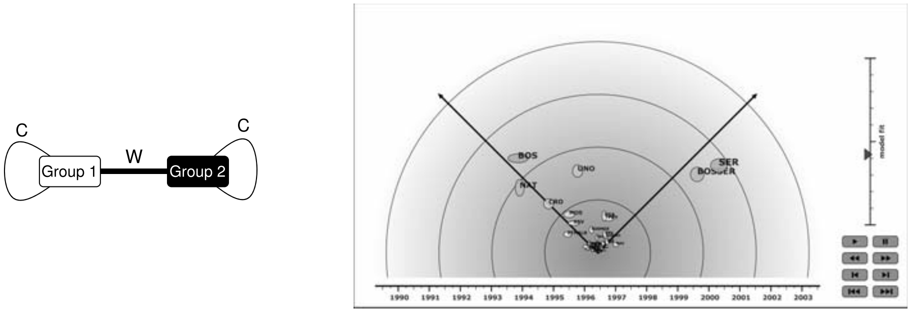


Fig. 5. *Left:* Quotient of a 2-dimensional conflict space. *Right:* Conflictive groups in the Balkans for the period from 1989 until 2003. Actors are mapped into the left(right) dimension to the extent that they are members of one of the two groups. The distance from the origin is a measure of how *involved* actors are in conflict.

$$c = \frac{\lambda + \mu}{2} \qquad \text{and} \qquad w = \frac{\lambda - \mu}{2} \ .$$

Theorem 2 implies that a similarity S is structural with $G/S = R_{cw}$, if and only if S is the projection onto the eigenvalues λ and μ of R_{cw}. Since our goal is to maximize the edge weight between the clusters, i. e., to maximize w, the optimal choice are the largest and the smallest eigenvalue of the adjacency matrix.

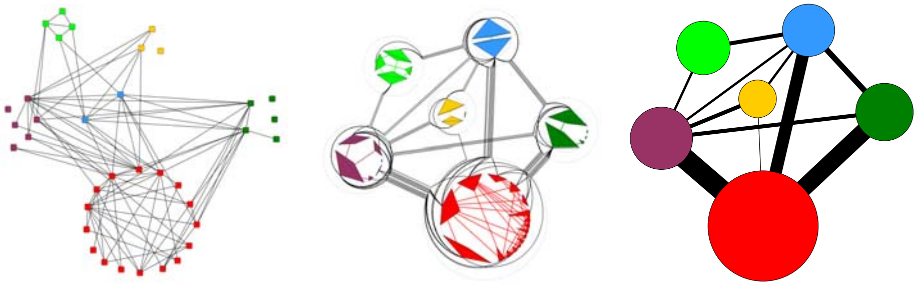
To obtain the actual degrees of membership to the two groups, the appropriate basis for the two-dimensional image space has to be identified. In short, the matrix P whose rows are the two eigenvectors has to be rotated by the inverse eigenvector-basis of R_{cw} (details can be found in [12]). An example for a projection to conflict space can be seen in Fig. 5 (*right*). As it can be seen, actors have largely differing degrees of membership and can also stand between groups. It has been shown in [15] how this method can be extended to more than two groups.

To show the development over time, we defined in [12] time-dependent conflict networks that take into account only the events within a certain time-frame. By letting this time-frame move forward, we obtain a smoothly animated visualization showing the development of conflicts over time.

3 Multi-circular Visualization

An important aspect in the visualization of many types of networks is the interplay between fine- and coarse-grained structures. While the micro-level graph is given, a macro-level graph is induced by a partitioning of the micro-level vertices. For example it may originate from a group-level network analysis such as a clustering or may just be given in advance.

We propose a tailored visualization for networks with such a micro/macro structure based on a novel multi-circular drawing convention. Given a layout of the macro-level graph with large nodes and thick edges, each vertex of the micro-level graph is drawn in the area defined by the macro-vertex it belongs to, and each micro-edge is routed through its corresponding macro-edge. In more detail, each micro-vertex is placed on a circle inside of the area of its corresponding macro-vertex and micro-edges whose end vertices belong to the same macro-vertex are drawn inside of these circles. All other micro-edges are then drawn inside of their corresponding macro-edges and at constant but different distances from the border of the macro-edge, i. e., in straight-line macro-edges they are drawn as parallel lines. These edges must also be routed inside the area of macro-vertices to connect to their endpoints, but are not allowed to cross the circles. Figure 6 shows a concrete example of this model. Micro-edges connecting vertices in the same macro-vertex are drawn as straight lines. Inside of macro-vertices, the other edges spiral around the circle of micro-vertices until they reach the area of the macro-edge. We give a combinatorial description of the above model and then focus on the algorithmically most challenging aspect of these layouts, namely crossing reduction by cyclic ordering of micro-vertices and choosing edge winding within macro-vertices.



(a) geometric grouping and straight-line edges

(b) multi-circular layout

(c) corresponding macro graph

Fig. 6. (a) Example organizational network with geometric grouping and straight-line edges (redrawn from [37]). In our multi-circular layout (b), all details are still present and the macro-structure induced by the grouping becomes clearly visible. Additionally, the height and width of the vertices reflects the number of connections within and between groups.

We do not impose restrictions on the macro-level layout other than sufficient thickness of edges and vertices, so that the micro-level graph can be placed on top of the macro-level graph, and provide layout algorithms and tailored means of interaction to support the generation of appropriate macro-layouts.

While the drawing convention consists of proven components—geometric grouping is used, e. g., in [37,50], and edge routing to indicate coarse-grained structure is proposed in, e. g., [6,34]—our approach is novel in the way micro-vertices are organized to let the macro-structure dominate the visual impression without cluttering the micro-level details too much. Note also that the setting is very different from layout algorithms operating on structure-induced clusterings (e. g., [3,36]), since no assumptions on the structure of clusters are made (they may even consist of isolates). Therefore, we neither want to utilize the clustering for a better layout, nor do we want to display the segregation into dense subregions or small cuts. Our aim is to represent the interplay between a (micro-level) graph and a (most likely extrinsic) grouping of its vertices.

After defining some basic terminology in Sect. 3.1, we state required properties for macro-graph layout in Sect. 3.2 and recapitulate related micro-layout models in Sect. 3.3. Multi-circular micro-graph layout is discussed in more detail in Sect. 3.4 and crossing reduction algorithms for it are given in Sect. 3.5.

3.1 Preliminaries

Throughout this section, we restrict ourselves to simple undirected graphs. In the following, let $E(v) = \{\{u,v\} \in E\,;\ u \in V\}$ denote the incident edges of a vertex $v \in V$, let $N(v) = \{u \in V\,;\ \{u,v\} \in E\}$ denote its neighbors, and let $\mathrm{sgn} : \mathbb{R} \to \{-1,0,1\}$ be the signum function.

Since each micro-vertex is required to belong to exactly one macro-vertex, the macro-structure defines a partition assignment $\rho : V \to \{1,\ldots,k\}$ and a prototypical macro-graph is the corresponding quotient graph $Q(G, \rho)$. An edge $\{u, v\} \in E$ is called an *intra-partition edge* if and only if $\rho(u) = \rho(v)$, and *inter-partition edge* otherwise. The set of intra-partition edges of a partition V_i is denoted by E_i, the set of inter-partition edges of two partitions V_i, V_j by $E_{i,j}$. We use $G = (V, E, \rho)$ to denote a graph $G = (V, E)$ and a related partition assignment ρ.

A *circular order* $\pi = \{\pi_1, \ldots, \pi_k\}$ defines for each partition V_i a vertex order π_i as a bijective function $\pi_i : V_i \rightarrow \{1, \ldots, |V_i|\}$ with $u \prec v \Leftrightarrow \pi_i(u) < \pi_i(v)$ for any two vertices $u, v \in V_i$. An order π_i can be interpreted as a counter-clockwise sequence of distinct positions on the circumference of a circle.

3.2 Macro Layout

No specific layout strategy for the macro-graph is required as long as its elements are rendered with sufficient thickness to draw the underlying micro-graph on top of them. In order to achieve this, post-processing can be applied to any given layout [29] or methods which consider vertex size (e. g., [33,53]) and edge thickness (e. g., [20]) have to be used.

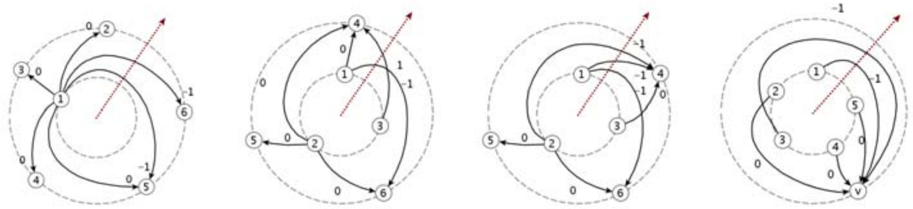
From a macro-layout we get *partition orders* $\Pi_i : N_Q(V_i) \rightarrow \{1, .., \deg(V_i)\}$ for each partition V_i, defined by the sequence of its incident edges in $Q(G, \rho)$, and a partition order $\Pi = \{\Pi_1, \ldots, \Pi_k\}$ for G. For each macro-vertex this can be seen as a counter-clockwise sequence of distinct docking positions for its incident macro-edges on its border.

3.3 Related (Micro) Layout

Before we discuss the multi-circular layout model for the micro-graph, let us recall the related concepts of (single) circular and radial embeddings. In *(single) circular layouts* all vertices are placed on a single circle and edges are drawn as straight lines. Therefore, a *(single) circular embedding* ε of a graph $G = (V, E)$ is fully defined by a vertex order π, i. e., $\varepsilon = \pi$ [8]. Two edges $e_1, e_2 \in E$ *cross* in ε if and only if the endvertices of e_1, e_2 are encountered alternately in a cyclic traversal.

In *radial level layouts* the partitions are placed on nested concentric circles (*levels*) and edges are drawn as curves between consecutive partitions. Therefore, only graphs $G = (V, E)$ with a *proper* partition assignment $\rho : V \rightarrow \{1, \ldots, k\}$ are allowed, i. e., $|\rho(u) - \rho(v)| = 1$ for all edges $\{u, v\} \in E$. Note that this prohibits intra-partition edges and edges connecting non-consecutive partitions. For technical reasons, edges are considered to be directed from lower to higher levels.

Recently, Bachmaier [5] investigated such layouts. They introduced a *ray* from the center to infinity to mark the start and end of the circular vertex orders. Using this ray, it is also possible to count how often and in which direction an edge is wound around the common center of the circles. We call this the *winding* $\psi : E \rightarrow \mathbb{Z}$ of an edge (Bachmaier called this *offset*). $|\psi(e)|$ counts the number of crossings of the edge with the ray and the sign reflects the mathematical direction



(a) some incident edges

(b) node 4 is at position 0

(c) node 4 rotated to position 2

(d) without parting

Fig. 7. Examples of Radial layouts. Edges are labeled with their winding value.

of rotation. See Fig. 7 for some illustrations. Finally, a *radial embedding* ε of a graph $G = (V, E, \rho)$ is defined to consist of a vertex order π and an edge winding ψ, i. e., $\varepsilon = (\pi, \psi)$.

There is additional freedom in radial drawings without changing the crossing number: the rotation of a partition V_i. A *rotation* moves a vertex v with extremal position in π_i over the ray. The layout in Fig. 7(c) is a clockwise rotation of the layout in Fig. 7(b). Rotations do not modify the cyclic order, i. e., the neighborhood of each vertex on its radial level is preserved. However, the winding of the edges incident to v and all positions of π_i must be updated.

Crossings between edges in radial embeddings depend on their winding and on the order of the endvertices. There can be more than one crossing between two edges if they have very different windings. The number of crossings between two edges $e_1, e_2 \in E$ in an radial embedding ε is denoted by $\chi_\varepsilon(e_1, e_2)$. The (radial) crossing number of an embedding ε and a level graph $G = (V, E, \rho)$ is then naturally defined as

$$\chi(\varepsilon) = \sum_{\{e_1,e_2\} \in E, e_1 \neq e_2} \chi_\varepsilon(e_1, e_2)$$

and $\chi(G) = \min\{\chi(\varepsilon) : \varepsilon \text{ is a radial embedding of } G\}$ is called the *radial crossing number* of G.

Theorem 4 ([5]). *Let $\varepsilon = (\pi, \psi)$ be a radial embedding of a two-level graph $G = (V_1 \dot{\cup} V_2, E, \rho)$. The number of crossings $\chi_\varepsilon(e_1, e_2)$ between two edges $e_1 = (u_1, v_1) \in E$ and $e_2 = (u_2, v_2) \in E$ is*

$$\chi_\varepsilon(e_1, e_2) = \max\left\{0, \left|\psi(e_2) - \psi(e_1) + \frac{b-a}{2}\right| + \frac{|a| + |b|}{2} - 1\right\} ,$$

where $a = \operatorname{sgn}(\pi_1(u_2) - \pi_1(u_1))$ and $b = \operatorname{sgn}(\pi_2(v_2) - \pi_2(v_1))$.

Bachmaier also states that in crossing minimal radial embeddings every pair of edges crosses at most once and adjacent edges do not cross at all. As a consequence, only embeddings need to be considered where there is a clear *parting* between all edges incident to the same vertex u. The parting is the position of the edge list of u that separates the two subsequences with different winding values. See again Fig. 7 for layouts with and without proper parting. Furthermore, only embeddings with small winding are considered because large winding values correspond to very long edges which are difficult to follow and generally result in more crossings.

3.4 Multi-circular Layout

Unless otherwise noted, vertices and edges belong to the micro-level in the following. In the micro-layout model each vertex is placed on a circle inside of its corresponding macro-vertex. Intra-partition edges are drawn within these circles as straight lines. Inter-partition edges are drawn inside their corresponding macro-edges and at constant but different distances from the border of the macro-edge. To connect to their incident vertices, these edges must also be routed inside of macro-vertices. Since they are not allowed to cross the circles, they are drawn as curves around them. Such a drawing is called a *(multi-)circular layout.* Since intra- and inter-partition edges cannot cross, all crossings of intra-partition edges are completely defined by the vertex order π_i of each partition V_i. Intuitively speaking, a vertex order defines a circular layout for the intra-partition edges. In the following we thus concentrate on inter-partition edges.

The layout inside each macro-vertex V_i can be seen as a two-level radial layout. The orders can be derived from the vertex order π_i and the partition order Π_i. Similar to radial layouts a *ray* for each partition is introduced and the beginning of the orders and the edge winding is defined according to these rays. Note that for each edge $e = \{u, v\} \in E$, $u \in V_i$, $v \in V_j$, two winding values are needed, one for the winding around partition V_i denoted by $\psi_i(e) = \psi_u(e)$, and one for the winding around partition V_j denoted by $\psi_j(e) = \psi_v(e)$. If the context implies an implicit direction of the edges, windings are called either source or target windings, respectively. Since radial layouts can be rotated without changing the embedding, rays of different partitions are independent and can be directed arbitrarily. Finally, a *multi-circular embedding* ε is defined by a vertex order π, a partition order Π, and the winding of the edges ψ, i. e., $\varepsilon = (\pi, \Pi, \psi)$.

Observation 5. *For each partition V_i in a multi-circular embedding ε=(π,Π,ψ) a two-level radial embedding $\varepsilon_i = ((\pi_i, \pi'), \psi_i)$ is defined by the vertex order π_i, the partition order Π_i, and the edge winding ψ_i, where $\pi'(v) = \Pi_i(\rho(v)), v \in V \setminus V_i$.*

There is another connection between radial and multi-circular layouts. A two-level radial layout can easily be transformed into a two-partition circular layout and vice versa. Given a graph $G = (V_1 \dot\cup V_2, E, \rho)$ and a radial embedding $\varepsilon = (\pi, \psi)$ of G, the two-partition circular embedding $\varepsilon^* = (\pi^*, \Pi^*, \psi^*)$ defined by $\pi_1^* = \pi_1$, $\pi_2^* = -\pi_2$, $\Pi_1^* = 0$, $\Pi_2^* = 0$, and $\psi_1^*(e) = \psi(e)$, $\psi_2^*(e) = 0$ realizes exactly the same crossings (see Fig. 8 for an example). Intuitively speaking, the topology of the given radial embedding is not changed if the two circles are dragged apart and one of the vertex orders is reversed. If a two-partition circular embedding $\varepsilon^* = (\pi^*, \Pi^*, \psi^*)$ is given, a related radial embedding $\varepsilon = (\pi, \psi)$ is defined by $\pi_1 = \pi_1^*$, $\pi_2 = -\pi_2^*$, and $\psi(e) = \psi_1(e) - \psi_2(e)$.

Observation 6. *There is a one-to-one correspondence between a two-level radial embedding and a two-circular embedding.*

Crossings in the micro-layout are due to either the circular embedding or crossing macro-edges. Since crossings of the second type cannot be avoided by changing the micro-layout, they are not considered in the micro-layout model. Obviously, pairs of edges which are not incident to a common macro-vertex can only cause crossings of this type. For pairs of edges which are incident to at least one common macro-vertex corresponding two-level radial layouts are defined using Observations 5 and 6 and the number of crossings are computed by modifications of Theorem 4.

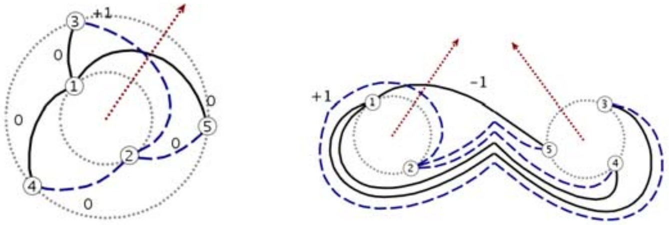


Fig. 8. A two-level radial layout and its corresponding two-circular layout

Theorem 7. *Let $\varepsilon = (\pi, \Pi, \psi)$ be a multi-circular embedding of a graph $G = (V, E, \rho)$ and let $e_1 = \{u_1, v_1\}$, $e_2 = \{u_2, v_2\} \in E$ be two inter-partition edges. If e_1 and e_2 share exactly one common incident macro-vertex, e. g., $V_i = \rho(u_1) = \rho(u_2)$, $\rho(v_1) \neq \rho(v_2)$, then the number of crossings of e_1 and e_2 is*

$$\chi_\varepsilon(e_1, e_2) = \max\left\{0, \left|\psi_i(e_2) - \psi_i(e_1) + \frac{b-a}{2}\right| + \frac{|a|+|b|}{2} - 1\right\},$$

where $a = \operatorname{sgn}(\pi_i(u_2) - \pi_i(u_1))$ and $b = \operatorname{sgn}(\Pi(\rho(v_2)) - \Pi(\rho(v_1)))$.

Proof. Let $e_1 = \{u_1, v_1\}$, $e_2 = \{u_2, v_2\} \in E$ be two edges with exactly one common end partition, e. g., $V_i = \rho(u_1) = \rho(u_2), \rho(v_1) \neq \rho(v_2)$. All crossings between e_1 and e_2 not caused by the macro layout occur in the macro-vertex V_i. According to Observation 5, the fraction of the layout in V_i can be regarded as a two-level radial layout defined by $\varepsilon' = (\pi_i, \Pi_i \circ \rho)$. Applying Theorem 4 to the embedding ε', the theorem follows. □

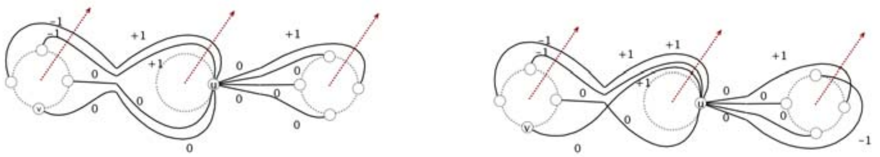
Theorem 8. *Let $\varepsilon = (\pi, \Pi, \psi)$ be a multi-circular embedding of a graph $G = (V, E, \rho)$ and let $e_1 = \{u_1, v_1\}$, $e_2 = \{u_2, v_2\} \in E$ be two inter-partition edges. If e_1 and e_2 belong to the same macro-edge, e. g., $V_i = \rho(u_1) = \rho(u_2)$, $V_j = \rho(v_1) = \rho(v_2)$, then the number of crossings between e_1 and e_2 is*

$$\chi_\varepsilon(e_1, e_2) = \max\left\{0, \left|\psi'(e_2) - \psi'(e_1) + \frac{b-a}{2}\right| + \frac{|a|+|b|}{2} - 1\right\} ,$$

where $a = \mathrm{sgn}(\pi_i(u_2) - \pi_i(u_1))$*,* $b = \mathrm{sgn}(\pi_j(v_1) - \pi_j(v_2))$ *and* $\psi'(e) = \psi_i(e) + \psi_j(e)$*.*

Proof. Let $e_1, e_2 \in E$ be two inter-partition edges which belong to the same macro-edge. Since only two partitions are involved, a two-level radial embedding ε' for e_1 and e_2 can be defined according to Observation 6. In ε' the two edges e_1 and e_2 cause the same crossings than in ε. Applying Theorem 4 to the embedding ε', the theorem follows. □

Similar to radial layouts, in a crossing minimal multi-circular embedding incident edges do not cross and there is at most one crossing between every pair of edges. Therefore, only embeddings need to be considered where there is a clear *parting* between all edges incident to the same vertex $u \in V_i$. Since in multi-circular layouts winding in different macro-vertices can be defined independently, the edge list $E(u)$ of u is split by target partitions resulting in edge lists $E(u)_j = \{\{u, v\} \in E(u) : v \in V_j\}$. For each list $E(u)_j$, a position ℓ_j separates the two subsequences with different values of winding ψ_j and defines the parting for this partition. Furthermore, there is also a parting for V_i defined on the edge list $E(u)$. The order of $E(u)$ for this parting depends on the partings ℓ_j in the target partitions V_j. Edges are sorted by the partition order and for edges to the same partition V_j, ties are broken by the reverse vertex order started not at the ray but at the parting position ℓ_j. Then, the parting for V_i is the position ℓ_i which separates different values of winding ψ_i in the so ordered list. See Fig. 9 for a layout with parting and a layout where the edge $\{u, v\}$ violates the parting.



(a) parting

(b) edge $\{u, v\}$ violates parting

Fig. 9. Not all winding combinations for the incident edges of u result in a good layout

Corollary 1. *Multi-circular crossing minimization is $\mathcal{NP}$-hard.*

Proof. Single circular and radial crossing minimization [5,43] are $\mathcal{NP}$-hard. As we have already seen, these two crossing minimization problems are subproblems of the multi-circular crossing minimization problem, proving the corollary.

As a consequence, we do not present exact algorithms for crossing minimization in multi-circular layouts. Instead, we propose extensions of some well-known crossing reduction heuristics for horizontal and radial crossing reduction.

3.5 Layout Algorithms

Since the drawing of inter-partition edges inside a macro-vertex can be seen as a radial drawing, a multi-circular layout can be composed of separate radial layouts for each macro-vertex (for instance using the techniques of [5,28,50]). However, such a decomposition approach is inappropriate since intra-partition edges are not considered at all and inter-partition edges are not handled adequately due to the lack of information about the layout at the adjacent macro-vertices. For example, choosing a path with more crossings in one macro-vertex can allow a routing with much less crossings on the other side.

Nevertheless, we initially present in this section adaptations of radial layout techniques because they are quite intuitive, fast, and simple, and can be used for the evaluation of more advanced algorithms.

Barycenter and Median Layouts. The basic idea of both the barycenter and the median layout heuristicis the following: each vertex is placed in a central location computed from the positions of its neighbors - in either the barycenter or the median position - to reduce edge lengths and hence the number of crossings. For a two-level radial layout, the *Cartesian Barycenter* heuristic gets the two levels and a fixed order for one of them. All vertices of the fixed level are set to equidistant positions on a circle and the component-wise barycenter for all vertices of the second level is computed. The cyclic order around the center defines the order of the vertices and the edges are routed along the geometrically shortest-path. The *Cartesian Median* heuristic is defined similar. Running time for both heuristics is in $\mathcal{O}(|E| + |V| \log |V|)$.

Both heuristics are easily extended for multi-circular layouts. The layout in each macro-vertex V_i is regarded as a separate two-level radial layout as described in Observation 6 and the partition orders Π_i are used to define the orders of the fixed levels. Because of the shortest-path routing, no two edges cross more than once and incident edges do not cross at all in the final layout. On the other hand, the used placement and winding strategies are based on edge length reduction and avoid crossings only indirectly.

Multi-circular Sifting. In order to overcome the drawbacks of the radial layout algorithms described before, we propose an extension of the sifting heuristic which computes a complete multi-circular layout and considers edge crossings for optimizing both vertex order and edge winding, and thus is expected to generate better layouts.

Sifting was originally introduced as a heuristic for vertex minimization in ordered binary decision diagrams [47] and later adapted for the layered one-sided, the circular, and the radial crossing minimization problems [5,8,44]. The idea is to keep track of the objective function while moving a vertex along a fixed order of all other vertices. The vertex is then placed in its (locally) optimal position. The method is thus an extension of the greedy-switch heuristic [21]. For crossing reduction the objective function is the number of crossings between the edges incident to the vertex under consideration and all other edges. In multi-circular layouts this function depends on both the vertex order and the edge winding. Therefore, for each position of a vertex, the winding values for its incident edges which result in the minimal crossing number have to be identified.

The efficient computation of crossing numbers in sifting for layered and single circular layouts is based on the locality of crossing changes, i. e., swapping consecutive vertices $u \curvearrowright v$ only affects crossings between edges incident to u with edges incident to v. In multi-circular layouts this property clearly holds for intra-partition edges since they form (single-)circular layouts. For inter-partition edges the best routing path may require an update of the windings. Such a change can affect crossings with all edges incident to the involved partitions.

Since swapping the positions of two consecutive vertices (and keeping the winding values) only affects incident edges, the resulting change in the number of crossings can be computed efficiently. Therefore, an efficient strategy for updating edge windings while $u \in V_i$ moves along the circle is needed. Instead of probing each possible combination of windings for each position of u the parting of the edge lists is considered. Note that the parting for the source partition and all the partings for the target partitions have to be simultaneously altered because for an edge, a changed winding in the source partition may allow a better routing with changed winding in the target partition. Intuitively speaking, the parting in the source partition should move around the circle in the same direction as u but on the opposite side of the circle, while the parting in the target partitions should move in the opposite direction. Otherwise, edge lengths increase and with them the likelihood of crossings. Thus, starting with winding values $\psi_u(e) = 1$ and $\psi_v(e) = 1$ for all $e = \{u, v\} \in E(v)$, parting counters are iteratively moved around the circles and mostly decreased in the following way:

1. First try to improve the parting at V_i, i. e., iteratively, the value of ψ_u for the current parting edge is decreased and the parting moves counter-clockwise to the next edge until this parting can no longer be improved.
2. For edges whose source winding are changed in step one, there may be better target windings which cannot be found in step three because the value of ψ_j has to be increased, i. e., for each affected edge, the value of ψ_j for the edge is increased until no improvement is made any more.
3. Finally try to improve the parting for each target partition V_j separately, i. e., for each V_j, the value of ψ_j for the current parting edge is decreased and the parting moves clockwise to the next edge until this parting can not be improved any further.

After each update, it is ensured that all counters are valid and that winding values are never increased above 1 and below -1.

Based on the above, the locally optimal position of a single vertex can be found by iteratively swapping the vertex with its neighbor and updating the edge winding while keeping track of the change in crossing number. After the vertex has passed each position, it is placed where the intermediary crossing counts reached their minimum. Repositioning each vertex once in this way is called a *round of sifting*.

Theorem 9. *The running time of multi-circular sifting is in $\mathcal{O}(|V||E|^2)$.*

Proof. Computing the difference in cross-count after swapping two vertices requires $\mathcal{O}(|E|^2)$ running time for one round of sifting. For each edge, the winding changes only a constant number of times because values are bounded, source winding and target winding are decreased in steps one and three, respectively, and the target winding is only increased for edges whose source winding decreased before. Counting the crossings of an edge after changing its winding takes time $\mathcal{O}(|E|)$ in the worst-case. Actually, only edges incident to the at most two involved macro-vertices have to be considered. For each vertex $u \in V$, the windings are updated $\mathcal{O}(|V| \cdot \deg(u))$ times, once per position and once per shifted parting. For one round, this results in $\mathcal{O}(|V||E|)$ winding changes. Together, the running time is in $\mathcal{O}(|V||E|^2)$. □

3.6 Example: Email Communication Network

The strength of a multi-circular layoutis the coherent drawing of vertices and edges at two levels of detail. It reveals structural properties of the macro-graph and allows identification of micro-level connections at the same time. The showcase for the benefits of our micro/macro layout is an email communication network of a department of the Universität Karlsruhe (TH). The micro-graph consists of 442 anonymized department members and 2 201 edges representing at least one email communication in the considered time frame of five weeks. At the macro-level, a grouping into 16 institutes is given, resulting in 66 macro-edges. In the following drawings, members of the same institute are colored identically.

We start by inspecting drawings generated by a general force-directed approach similar to the method of Fruchterman and Reingold [26] and by multidimensional scaling (MDS) [17], see Fig. 10. Both methods tend to place adjacent vertices near each other but ignore the additional grouping information. Therefore, it is not surprising that the drawings do not show a geometric clustering and the macro-structure cannot be identified. Moreover, it is difficult or even impossible to follow edges since they massively overlap each other.

More tailored for the drawing of graphs with additional vertex grouping are the layout used by Krebs [37], and the force-directed attempts to assign vertex positions by Six and Tollis [50] and Krempel [38]. All three methods place the vertices of each group on circles inside of separated geometric areas. While some efforts are made to find good vertex positions on the circles, edges are simply drawn as straight lines. Figure 6(a) gives a prototypical example of this layout style. Although these methods feature a substantial progress compared to general layouts and macro-vertices are clearly visible, there is no representation of macro-edges and so the overall macro-structure is still not identifiable.

Finally, we investigate multi-circular visualizations of the email network. Its combinatorial descriptions allows for enrichments with analytical visualizations of the vertices. In Fig. 11 the angular width of the circular arc a vertex covers is proportional to its share of the total inter-partition edges of this group. The height from its chord to the center of the circle reflects the fraction of present to possible intra-edges.

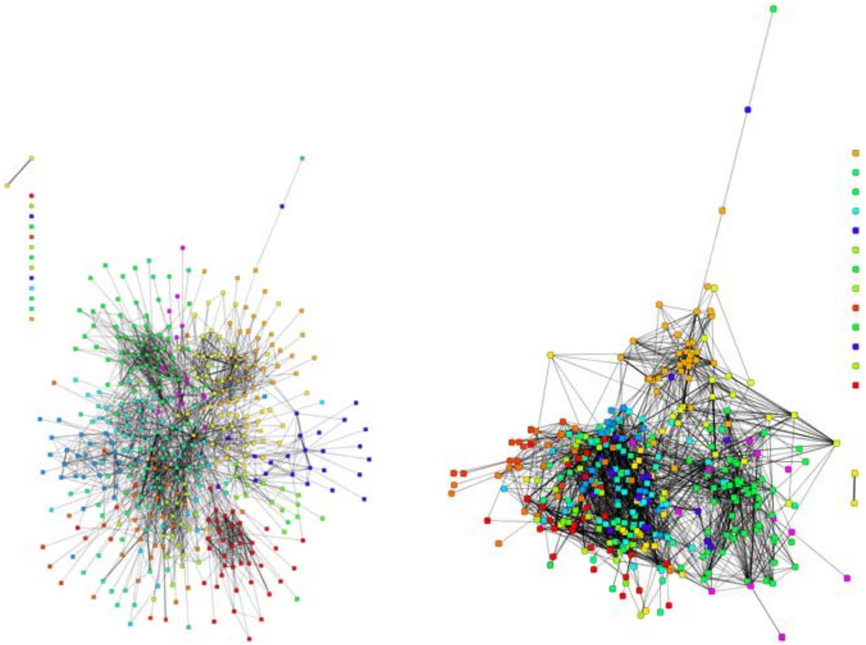
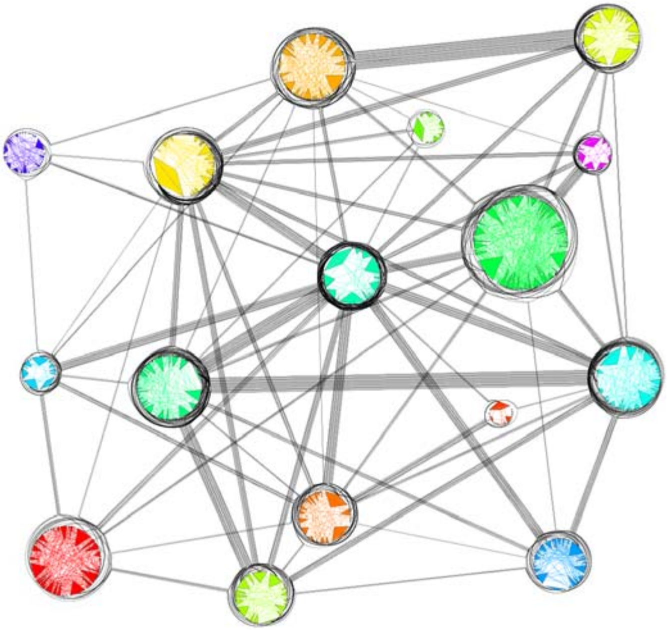


Fig. 10. Drawings of the email network generated by a force-directed method (left) and by multidimensional scaling (MDS, right). The colors of the vertices depict the affiliation to institutes.

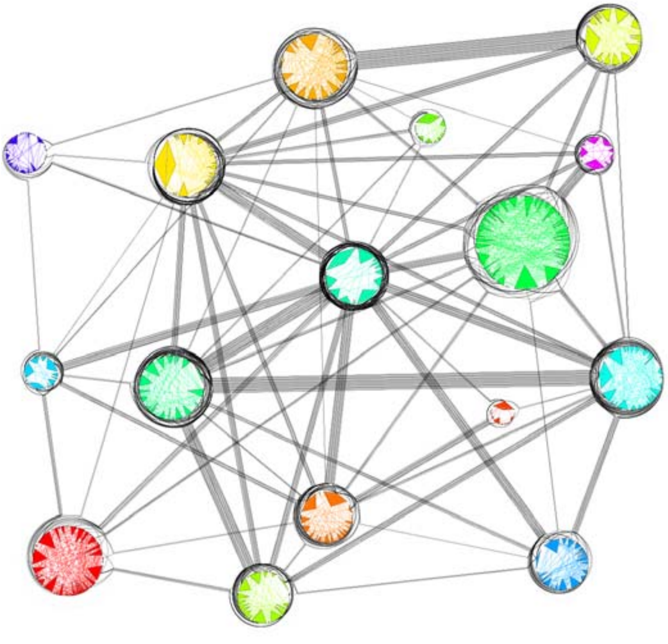
In order to investigate the effect of improved vertex orders and appropriate edge windings, we compare three variations of drawing heuristics for multi-circular layouts: shortest-path edge winding combined with random vertex placement and with barycenter vertex placement, and our multi-circular sifting (see Fig. 11). Through the grouping of micro-edges the macro-structure of the graph is apparent at first sight. A closer look reveals the drawback of random placement: edges between different groups have to cover a long distance around the vertex circles and are hard to follow. Also a lot of edge crossings are generated both inside of the groups and in the area around the vertex placement circles. Assigning vertex positions according to the barycenter heuristic results in a clearly visible improvement and allows the differentiation of some of the micro-edges. Using sifting improves the layout even further, resulting from a decrease of the number of crossings from more than 75 000 to 57 400 in the considered email network. The time for computing the layout of this quiet large graph is below 10 seconds.

3.7 Final Remarks

From the micro-layout algorithm we get a combinatorial description of the layout, i. e., circular vertex orders for each partition and edge windings, which allows



(a) barycenter (68 300 crossings)



(b) sifting (57 400 crossings)

Fig. 11. Multi-circular layouts of the email network

arbitrarily rotating each circular order without introducing new crossings (see Section 3.4). Therefore, for each partition, a rotation is chosen which minimizes the total angular span of the inter-partition edges, reserve space for the drawing of these edges, and place the vertices accordingly at a uniform distance from the border of the macro-vertex.

A major benefit of the multi-circular layout is its combinatorial description since it allows the combination with other visualization techniques to highlight some graph properties or to further improve the visual appearance.

4 Conclusion

In this paper we presented methods to analyze and visualize group-structure in social networks. The first part is focused on the definition and computation of structural network positions. We started by reviewing previous approaches for this task and distinguished them by two different criteria: first, whether the method establishes equivalence of actors or similarity of actors (discrete vs. real valued approaches) and, second, whether similarity is based on neighborhood identity or neighborhood equivalence. We then presented a novel framework for defining and computing network positions that is general and enjoys several methodological advantages over previous approaches. The second part of this paper introduced a visualization technique that shows the interplay between fine-grained (individual level) and coarse-grained (group level) network structure. Special emphasis has been given to the algorithmic optimization of esthetic criteria such as reducing the number of edge crossings.

References

1. Achlioptas, D., Fiat, A., Karlin, A., McSherry, F.: Web Search Via Hub Synthesis. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2001), pp. 500–509 (2001)
2. Alon, N., Kahale, N.: A Spectral Technique for Coloring Random 3-Colorable Graphs. SIAM Journal on Computation 26, 1733–1748 (1997)
3. Archambault, D., Munzner, T., Auber, D.: TopoLayout: Multi-Level Graph Layout by Topological Features. IEEE Transactions on Visualization and Computer Graphics 13(2), 305–317 (2007)
4. Azar, Y., Fiat, A., Karlin, A., McSherry, F., Saia, J.: Spectral Analysis of Data. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 619–626 (2001)
5. Bachmaier, C.: A Radial Adaptation of the Sugiyama Framework for Visualizing Hierarchical Information. IEEE Transactions on Visualization and Computer Graphics 13(3), 585–594 (2007)
6. Balzer, M., Deussen, O.: Level-of-Detail Visualization of Clustered Graph Layouts. In: Asia-Pacific Symposium on Visualisation 2007, APVIS 2007 (2007)
7. Batagelj, V., Doreian, P., Ferligoj, A.: An Optimizational Approach to Regular Equivalence. Social Networks 14, 121–135 (1992)

8. Baur, M., Brandes, U.: Crossing Reduction in Circular Layouts. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 332–343. Springer, Heidelberg (2004)
9. Borgatti, S.P., Everett, M.G.: Notions of Position in Social Network Analysis. Sociological Methodology 22, 1–35 (1992)
10. Brandes, U.: On Variants of Shortest-Path Betweenness Centrality and their Generic Computation. Social Networks 30(2), 136–145 (2008)
11. Brandes, U., Erlebach, T. (eds.): Network Analysis: Methodological Foundations. Springer, Heidelberg (2005)
12. Brandes, U., Fleischer, D., Lerner, J.: Summarizing Dynamic Bipolar Conflict Structures. IEEE Transactions on Visualization and Computer Graphics, special issue on Visual Analytics 12(6), 1486–1499 (2006)
13. Brandes, U., Lerner, J.: Structural Similarity in Graphs. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 184–195. Springer, Heidelberg (2004)
14. Brandes, U., Lerner, J.: Coloring Random 3-Colorable Graphs with Non-uniform Edge Probabilities. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 202–213. Springer, Heidelberg (2006)
15. Brandes, U., Lerner, J.: Visualization of Conflict Networks. In: Kauffmann, M. (ed.) Building and Using Datasets on Armed Conflicts. NATO Science for Peace and Security Series E: Human and Societal Dynamics, vol. 36. IOS Press, Amsterdam (2008)
16. Coja-Oghlan, A.: A Spectral Heuristic for Bisecting Random Graphs. In: Proceeding of the 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 850–859 (2005)
17. Cox, T.F., Cox, M.A.A.: Multidimensional Scaling, 2nd edn. Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, Boca Raton (2001)
18. Cvetković, D.M., Doob, M., Sachs, H.: Spectra of Graphs. Johann Ambrosius Barth Verlag (1995)
19. Doreian, P., Batagelj, V., Ferligoj, A.: Generalized Blockmodeling. Structural Analysis in the Social Sciences, vol. 25. Cambridge University Press, Cambridge (2005)
20. Duncan, C.A., Efrat, A., Kobourov, S.G., Wenk, C.: Drawing with Fat Edges. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 162–177. Springer, Heidelberg (2002)
21. Eades, P., Kelly, D.: Heuristics for Reducing Crossings in 2-Layered Networks. Ars Combinatoria 21(A), 89–98 (1986)
22. Everett, M.G., Borgatti, S.P.: Regular Equivalence: General Theory. Journal of Mathematical Sociology 18(1), 29–52 (1994)
23. Fiala, J., Paulusma, D.: The Computational Complexity of the Role Assignment Problem. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 817–828. Springer, Heidelberg (2003)
24. Fleischer, D.: Theory and Applications of the Laplacian. Ph.D thesis (2007)
25. Freeman, L.C.: Visualizing Social Networks. Journal of Social Structure 1(1) (2000)
26. Fruchterman, T.M.J., Reingold, E.M.: Graph Drawing by Force-Directed Placement. Software - Practice and Experience 21(11), 1129–1164 (1991)
27. Gaertler, M.: Algorithmic Aspects of Clustering – Theory, Experimental Evaluation and Applications in Network Analysis and Visualization. Ph.D thesis, Universität Karlsruhe (TH), Fakultät für Informatik (2007)
28. Gansner, E.R., Koren, Y.: Improved Circular Layouts. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 386–398. Springer, Heidelberg (2007)
29. Gansner, E.R., North, S.C.: Improved Force-Directed Layouts. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 364–373. Springer, Heidelberg (1999)

30. Gkantsidis, C., Mihail, M., Zegura, E.W.: Spectral Analysis of Internet Topologies. In: Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom), vol. 1, pp. 364–374. IEEE Computer Society Press, Los Alamitos (2003)
31. Godsil, C., Royle, G.: Algebraic Graph Theory. Graduate Texts in Mathematics. Springer, Heidelberg (2001)
32. Golub, G.H., van Loan, C.F.: Matrix Computations. John Hopkins University Press, Baltimore (1996)
33. Harel, D., Koren, Y.: Drawing Graphs with Non-Uniform Vertices. In: Proceedings of the 6th Working Conference on Advanced Visual Interfaces (AVI 2002), pp. 157–166. ACM Press, New York (2002)
34. Holten, D.: Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. IEEE Transactions on Visualization and Computer Graphics 12(5), 741–748 (2006)
35. Kannan, R., Vempala, S., Vetta, A.: On clusterings: Good, Bad and Spectral. Journal of the ACM 51(3), 497–515 (2004)
36. Kaufmann, M., Wiese, R.: Maintaining the Mental Map for Circular Drawings. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 12–22. Springer, Heidelberg (2002)
37. Krebs, V.E.: Visualizing Human Networks. Release 1.0, pp. 1–25 (1996)
38. Krempel, L.: Visualisierung komplexer Strukturen. Grundlagen der Darstellung mehrdimensionaler Netzwerke. Campus (2005)
39. Lerner, J.: Role Assignments. In: Brandes, U., Erlebach, T. (eds.) Network Analysis. LNCS, vol. 3418, pp. 216–252. Springer, Heidelberg (2005)
40. Lerner, J.: Structural Similarity of Vertices in Networks. Ph.D thesis (2007)
41. Lorrain, F., White, H.C.: Structural Equivalence of Individuals in Social Networks. Journal of Mathematical Sociology 1, 49–80 (1971)
42. Luczkovich, J.J., Borgatti, S.P., Johnson, J.C., Everett, M.G.: Defining and Measuring Trophic Role Similarity in Food Webs Using Regular Equivalence. Journal of Theoretical Biology 220(3), 303–321 (2003)
43. Masuda, S., Kashiwabara, T., Nakajima, K., Fujisawa, T.: On the $\mathcal{NP}$-Completeness of a Computer Network Layout Problem. In: Proceedings of the 20th IEEE International Symposium on Circuits and Systems 1987, pp. 292–295. IEEE Computer Society, Los Alamitos (1987)
44. Matuszewski, C., Schönfeld, R., Molitor, P.: Using Sifting for k-Layer Straightline Crossing Minimization. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 217–224. Springer, Heidelberg (1999)
45. McSherry, F.: Spectral Partitioning of Random Graphs. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2001), pp. 529–537 (2001)
46. Papadimitriou, C., Raghavan, P., Tamaki, H., Vempala, S.: Latent Semantic Indexing: A Probabilistic Analysis. Journal of Computer and System Sciences 61(2), 217–235 (2000)
47. Rudell, R.: Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In: Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, pp. 42–47. IEEE Computer Society Press, Los Alamitos (1993)
48. Sailer, L.D.: Structural Equivalence: Meaning and Definition, Computation and Application. Social Networks 1, 73–90 (1978)
49. Schrodt, P.A., Davis, S.G., Weddle, J.L.: Political Science: KEDS - A Program for the Machine Coding of Event Data. Social Science Computer Review 12(3), 561–588 (1994)

50. Six, J.M., Tollis, I.G.: A Framework for User-Grouped Circular Drawings. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 135–146. Springer, Heidelberg (2004)
51. Stewart, G.W., Sun, J.-G.: Matrix Perturbation Theory. Academic Press, London (1990)
52. Vempala, S., Wang, G.: A Spectral Algorithm for Learning Mixtures of Distributions. In: Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2002 (2002)
53. Wang, X., Miyamoto, I.: Generating Customized Layouts. In: Brandenburg, F.J. (ed.) GD 1995. LNCS, vol. 1027, pp. 504–515. Springer, Heidelberg (1996)
54. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press, Cambridge (1994)
55. White, D.R., Reitz, K.P.: Graph and Semigroup Homomorphisms on Networks of Relations. Social Networks 5, 193–234 (1983)

Modeling and Designing Real–World Networks

Michael Kaufmann[1] and Katharina Zweig[2]

[1] University of Tübingen,
Wilhelm–Schickard–Institute für Informatik,
Sand 14, 72076 Tübingen, Germany
mk@informatik.uni-tuebingen.de
[2] Eötvös Loránd University (ELTE)
Department of Biological Physics,
Pazmany P. stny 1A,
1117 Budapest, Hungary
nina@angel.elte.hu

Abstract. In the last 10 years a new interest in so–called real–world graph structures has developed. Since the structure of a network is crucial for the processes on top of it a well–defined network model is needed for simulations and other kinds of experiments. Thus, given an observable network structure, models try to explain how they could have evolved. But sometimes also the opposite question is important: given a system with specific constraints what kind of rules will lead to a network with the specified structure? This overview article discusses first different real–world networks and their structures that have been analyzed in the last decade and models that explain how these structures can emerge. This chapter concentrates on those structures and models that are very simple and can likely be included into technical networks such as P2P-networks or sensor networks. In the second part we will then discuss how difficult it is to design local network generating rules that lead to a globally satisfying network structure.

1 Introduction

Since the 1950s a considerable part of graph theory was devoted to the study of theoretical graph models, so–called random graphs that are created by a random process. Especially the random graph models $G(n, m)$ defined by Erdös and Rényi in 1959 [19] and the $G(n, p)$-model introduced by Gilbert [22] proved themselves to be very handy and well analyzable [9]. Other simple graph classes like regular graphs, grid graphs, planar graphs, and hypercubes were also defined and analyzed, and many of their properties proved to be useful in algorithm design, making hard problems considerable easier. It is reasonable that most real–world networks neither belong to any of the simple graph classes nor that they can be fully modeled as random graphs. But on the other hand, it seemed to be reasonable that at least some real-world networks can be considered to be random on a global scale, for example the so–called weblink graph: the weblink graph represents webpages as vertices and connects two vertices with a (directed)

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 359–379, 2009.

edge when the first page links to the second. Although it can be assumed that most of the time similar pages link to each other, the vast number of single decisions could be assumed to randomize this structure on a global scale. Thus, it came with a surprise when in 1999, Barabási and Albert showed that the real weblink graph cannot be modeled by a random graph since most of the webpages have only a low degree of in- and outcoming links while some have a huge degree, e.g., Yahoo or Google [6]. This phenomenon could not be explained by any of the classic random graph models. Another finding by Watts and Strogatz showed that real–world networks combine two properties that are not captured by any of the classic network models: real–world networks tend to be clustered, i.e., the neighbors of any vertex are likely to be connected, similar to grid graphs, but at the same time the average distance between all the vertices is much lower than expected in such a grid–like graph and resembles that of a random graph [48].

These findings opened a new field in between empirical sciences, mostly physics, biology, and the social sciences, and theoretical sciences as computer science and mathematics. This field is called *complex network science* [5,38][1]. In a very broad definition, we will denote by *complex network science* all research that can be subsumed under the following three perspectives:

1. **Complex Network Analysis:** measures and algorithms introduced to understand the special structure of real–world networks by differentiating them from established graph models.
2. **Complex Network Models:** models that capture essential structural properties of real–world networks and algorithms that construct them.
3. **Processes on Complex Networks:** analysis of the outcome of a process or algorithm on a given network structure.

In the following we will summarize the progress in the field of complex network models and show its relevance for algorithm design and computer simulations. Section 2 gives the necessary definitions and Section 3 discusses two principally different ways of modeling real–world network data, the *data–driven* and the *mechanistic* approach. Section 4 focuses on different mechanistic network models for real–world networks. Section 5 discusses some of the difficulties in designing networks for a set of independent agents. A summary and discussion of open problems is given in Section 6. Section 7 gives a list of well-organized resources for further reading.

2 Definitions

A graph G is a pair of sets (V, E) where $V = \{v_1, \ldots, v_n\}$ denotes a set of vertices and $E \subseteq V \times V$ denotes a *relation* between these vertices. If all edges of a graph are given as unordered pairs of vertices, the graph is said to be *undirected.* The degree $deg(v)$ of vertex v is defined as the number of edges it is element of.

[1] Complex network science strongly overlaps with a new field called *web science*, introduced by (among others) Sir Berners-Lee [8].

We will denote undirected edges e by pairs of vertices in simple brackets (v, w). A *weight function* $\omega : E \rightarrow \mathbb{R}$ assigns a weight to each edge. If the graph is unweighted, it is convenient to set $\omega(e) := 1$ for all edges. A *path* $P(s, t)$ from vertex s to vertex t is an ordered set of consecutive edges $\{e_1, e_2, \ldots, e_k\} \subseteq E$ with $e_1 = (s, v_1), e_k = (v_{k-1}, t)$ and $e_i = (v_{i-1}, v_i)$, for all $1 < i < k$. The *length of a path* $l(P(s, t))$ is defined as the sum over the weights of the edges in the path:

$$l(P(s,t)) = \sum_{e \in P(s,t)} \omega(e). \tag{1}$$

A path $P(s, t)$ is a *shortest path* between s and t if it has minimal length of all possible paths between s and t. The *distance* $d(s, t)$ between s and t is defined as the length of a shortest path between them. If there is no path between any two vertices, their distance is ∞ by definition. The *diameter* of a given graph is defined as the maximal distance between any two of its vertices if the graph is connected and defined to be ∞ if it is unconnected.

We will denote as a *real–world network* or, synonymously, as a *complex network* any network that presents an abstract view on a complex system, i.e., a real–world system comprised of different kinds of objects and relations between them. A *complex network* depicts normally only one class of objects and one kind of relation between the instances of this object class. Examples for complex networks are the weblink graph, that depicts web pages and links connecting them, social networks, e.g., the hierarchical structures between employers of a company, or transport networks, i.e., certain places that are connected by means of transportation like streets or tracks. Thus, real–world or complex networks do not comprise a new kind of graph class that can structurally be differentiated from other graph classes. The term merely denotes those graphs that represent at least one real–world system in the above sense.

One of the structural properties of a graph is its *degree distribution*, i.e., the number of vertices with degree $deg(v) = k$ in dependence of the degree. It is well known that random graphs have a Poissonian degree distribution [9], with a mean degree of np and standard deviation of $\sqrt{np}$.

A *graph family* $\mathcal{G}_A(n, \Pi)$ is a set or graph defined by some algorithm A that gives a description to construct graphs for every given n and - if needed - an additional set of parameters Π. If A is a *deterministic algorithm* it constructs a single graph, if it is a *stochastic algorithm* it constructs all graphs with n vertices (and maybe additional parameters specified by Π) with a determined probability. The instance that is created from some defined graph family $\mathcal{G}_A(n, \Pi)$ is denoted by $G_A(n, \Pi)$. For graph families, we will often state expected properties with the words: *with high probability*, denoting that an instance $G_A(n, \Pi)$ constructed by A will show property X with a probability higher than $1 - 1/n$.

A *random graph* $G(n, p)$ is a graph with n vertices where every (undirected or directed) edge is element of E with probability p. A different but related algorithm for constructing random graphs is the $G(n, m)$ family of random graphs that picks m pairs of vertices and connects them with each other. Most of the time an implementation will try to avoid self-loops and multiple edges. Bollobás states that for $\lim n \to \infty$ all expected properties of both families will be the same [9]. In the following, the term *random graph* will always denote an instance from the $G(n, p)$ model.

3 Data–Driven, Mechanistic, and Game Theoretic Network Models

There are principally different ways of modeling real–world network data: one way tries to model one data set as closely as possible, resulting in a very accurate model of a given system. We call this the *data–driven approach* to network modeling. The other way searches for a structure that is common in many different systems, a so–called *universal structure*, and tries to explain the emergence of this structure with as few parameters as possible. These models follow the *mechanistic approach*. A third approach that is also often called a *network formation game* models complex systems in which networks emerge between independent agents. In most of these models the mechanism by which individual agents form bonds is given and the resulting network is in the focus of the analysis. In the following we will discuss these three types of network models and argue why and how mechanistic networks can be helpful in computer science.

Given an empirical data set, e.g., a social network that is explored by giving questionnaires to a group of people, it is necessary to model it as closely as possible while allowing some randomness. The random element allows for missing or false data which is often a problem in survey–based data and it also allows for deducing from the model the general behavior of other social networks with a similar overall structure. E.g., by exploring one email contact network in one company general statements might be possible about email contact networks in other companies with the same structure. These kind of graph models that try to describe a given empirical data set as closely and with as little parameters as possible can be called *data–driven graph models*. A very popular approach of this kind is the *exponential random graph model* [42,43] and *block modeling* [39].

A very different perspective on modeling originates in the physics community: instead of modeling the details of a given graph they try to find an essential, mechanistic model that explains how a given structure could emerge in many different systems. These models are rather simplistic and will thus not provide very realistic models for any specific data set. But since they are so simplistic they can often be very easily adjusted to a specific system by adding back the peculiarities of that system.

Note that game–theoretic models of network formation are also very popular, but their perspective is in a way opposite to that of data–driven and mechanistic network models [7]: they define some kind of mechanism that determines how so–called agents decide which edges to build. The main question of these models is then which kind of network structures are stable in the sense that none of the agents would prefer to alter its own edge set. Thus, whereas data–driven and mechanistic network models start from a given structure that is modeled, game–theoretic approaches start with a presumed network formation mechanism and analyze the resulting network structure.

In this chapter we will concentrate on *mechanistic graph models* that explain universal network structures with the least number of parameters necessary to produce them. Since these models are so simple but the resulting networks still show some very helpful properties, they can easily be adapted to be used in different computer–aided networks, e.g., P2P– or sensor–networks. We will thus concentrate on these mechanistic models in the rest of the chapter. We start by discussing the most common universal network structures and sketching some of the simple, mechanistic models that describe a reasonable mechanism for their emergence.

4 Real–World Network Structures and Their Mechanistic Network Models

Among the structures found so far, the following four seem to be quite universal in many kinds of complex networks:

1. Almost all complex networks are *small–worlds*, i.e., locally they are densely connected while the whole graph shows a small diameter [48] (s. Subsect. 4.1);
2. Most of all complex networks are *scale–free*, i.e., most vertices have a low degree but some have a very high degree [6] (s. Subsect. 4.2);
3. Many of them are *clustered*, i.e., it is possible to partition the graph into groups of dense subgraphs whose interconnections are only sparse [16,23,37,40];
4. At least some of them seem to be *fractal* [44].

Other structures, especially small subgraphs in directed networks, so–called *network motifs*, have been identified in only a few networks, especially biological networks [35,36]. We will concentrate on the first two structures since it has already been shown that these properties influence processes on networks that are relevant in computer science as we will sketch in the following.

4.1 Small–Worlds

In 1998, Watts and Strogatz reported that in many real–world networks the vertices are locally highly connected, i.e., *clustered*, while they also show a small average distance to each other [48]. To measure the clustering they introduced the so–called *clustering coefficient* $cc(v)$ of a vertex v to be:

$$cc(v) = \frac{2e(v)}{deg(v)(deg(v)-1)}, \tag{2}$$

where $e(v)$ denotes the number of edges between all neighbors of v[2]. Since the denominator gives the possible number of those edges, the clustering coefficient of a single vertex denotes the probability that two of its neighbors are also connected by an edge. The *clustering coefficient* $CC(G)$ of a graph is the average

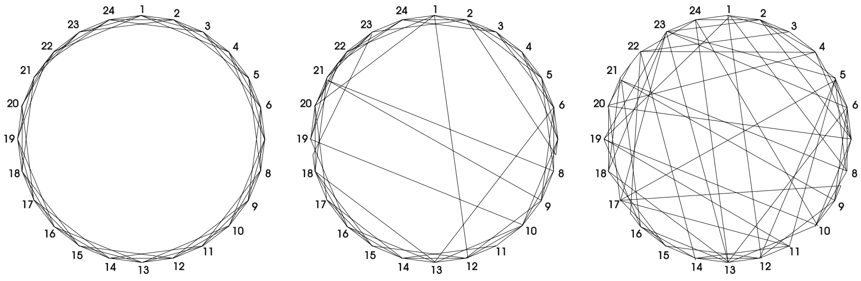
[2] The clustering coefficient of vertices with degree 1 is set to 0 and it is assumed that the graph does not contain isolated vertices.

over all the clustering coefficients of its vertices. A high average clustering coefficient in a graph seems to indicate that the vertices are connected locally and that thus the average distance of the according graph will be large. This is, e.g., the case in a simple grid, where vertices are only connected to their next four neighbors and thus the diameter scales with $\sqrt{n}$. Astonishingly, the diameter of the real–world networks analyzed in the paper of Watts and Strogatz was more similar to that of a corresponding random graph from the $G(n, p)$ model, despite their high average clustering coefficients. A random graph is said to be *corresponding* to a real–world network if it has the same number of vertices and expectedly the same number of edges. Such a graph can be achieved by setting p to $2m/(n(n-1))$. Of course, the expected clustering coefficient of vertices in such a graph will be p since the probability that any two neighbors of vertex v are connected is p. It now turned out that the real–world networks the authors analyzed had a clustering coefficient that was up to $1,000$ times higher than that of a corresponding random graph.

The first mechanistic model to reproduce this behavior, i.e., a high average clustering coefficient combined with small average distance, was given by Watts and Strogatz (s. Fig. 1): They start with a set of n vertices in a circular order where each vertex is connected with an undirected edge to its k clockwise next neighbors in the order. Such a graph is also called a *circulant graph.* Each edge is subsequently *rewired* with probability $0 \leq p \leq 1$. To *rewire* an edge $e = (v, w)$ a new target vertex w' is chosen uniformly at random, and (v, w) is replaced by (v, w'). It is clear that with $p = 0$, the clustering coefficient is given by:

$$CC(G) = \frac{3(k-1)}{2(2k-1)}, \qquad [38, p.289] \tag{3}$$

which approaches $3/4$ in the limit of large k. The average distance between pairs of vertices is $n/4k$ which scales linearly with the number of vertices. If now the average clustering coefficient and the average distance is analyzed with respect to p (s. Fig. 2), it is easy to see that the average distance drops much faster



(a) $G_c(24, 3)$, $p = 0$ (b) $p = 0.1$ (c) $p = 1$

Fig. 1. a) A circulant graph of 24 vertices where each vertex is connected to its three nearest neighbors. **b)** Each edge has been rewired with probability $p = 0.1$. **c)** Each edge has been rewired with $p = 1.0$.

than the average clustering coefficient. Thus, for small p, the average clustering coefficient is still quite high while the average distance has already dropped to a value comparable to that of a corresponding random graph. The regime in which that happens defines the set of networks showing the small–world effect or, for short: the set of small-world networks. Note that this very blurry definition has never been stated more rigorously. As Newman, Watts, and Barabási put it, nowadays the term "small–world effect" has come to mean that the average distance in the network increases at most (poly–)logarithmically with n while the average clustering coefficient is much higher than $p = 2m/(n(n-1))$.

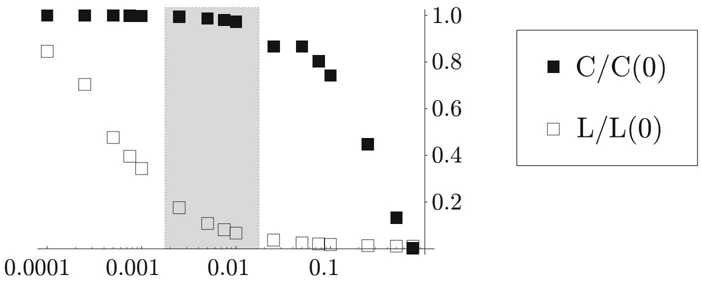


Fig. 2. Shown is the average clustering coefficient (denoted by C here) and the average distance (L) in the Watts–Strogatz–model, in dependence of the rewiring probability p and normalized by the resulting value for $p = 0$ ($C(0)$, $L(0)$). The average clustering coefficient is quite stable as long as not too many edges are rewired, while the average distance drops very fast, even at very low values of p.

Next to this rewiring model, other authors suggested to model small–world networks by composing something like a local, grid–like graph with a very sparse random graph, e.g., Kleinberg [26,25] and Andersen et al. in their hybrid–graph model [4,14]. The simplest model is to take a d–dimensional grid graph on n vertices and additionally connect each pair of vertices with probability p. This hybrid graph model is denoted by $G_d(n, p)$. Of course, if p is around $(\log n)^{1+\epsilon}/n$ for some constant $\epsilon > 0$ the edges constituting the random graph part alone will induce an average distance of $O(\log n)$ [10]. As we could show, also much smaller values of p suffice to reduce the diameter (and thus the average distance) to a poly-logarithmical term:

Lemma 1. *[29,49] For $p = \frac{1}{cn}$, $c \in \mathbb{R}^+$ and $\epsilon > 0$ the diameter of $G_d(n, p)$ is asymptotically bound with high probability by at most*

$$d \cdot \left(\left\lceil \sqrt[d]{c \cdot (\log n)^{1+\epsilon}} \right\rceil - 1 \right) \cdot \left(\frac{\log n}{(1+\epsilon) \log\log n - \log 2} + 1 \right). \qquad (4)$$

Very broadly speaking: the diameter of a combined graph is linearly dependent on the dimension d of the underlying grid and on (very broadly) $O(\log n^{1+(1+\epsilon)/d})$.

This result can be generalized as follows:

Lemma 2. *[29,49] For any function* $(\log n)^{-(1+\epsilon)} \leq f(n) \leq n^{1-\delta}, \epsilon, \delta > 0$ *and* $p = \frac{1}{f(n)\cdot n}$ *the diameter of* $G_d(n,p)$ *approaches asymptotically with high probability*

$$d \cdot \left(\left\lceil \sqrt[d]{f(n) \cdot (\log n)^{1+\epsilon}} \right\rceil - 1 \right) \cdot \left(\frac{\log(n/f(n))}{\log(\log n)^{1+\epsilon}} \right). \tag{5}$$

Roughly speaking, if p is set to $1/(n \log n)$, i.e., only every $\log n$-th vertex is incident with a random edge, the average distance in the $G_d(n,p)$ model for large n and $d = 2$ is given by $O(\sqrt{\log^{2+\epsilon} n} \log n)$, i.e., $O(\log^{2+\epsilon/2} n)$. This result is important for network design: let us assume that the cost for connecting two communication devices scales with their distance. If only connections up to a certain distance are built, the diameter of the resulting network will scale approximately linearly with the largest distance. Our result now shows that only a small amount of random–like edges has to be built in order to shrink the network's diameter to a quadratic logarithmic term.

It could be shown that small–worlds are ubiquitious. Especially important for computer scientists, all technical communication networks, like the Internet and various overlay–networks, have the property that they show a small average distance together with a high clusteredness. Thus, if a new kind of Internet protocol or a new P2P–system is simulated, it is very important to simulate them on a network model that comprises these two properties.

As stated above, the small–world effect assumes that the average distance is bound by $O(\log^k n)$ for some constant k. The models given above will essentially show a growing average distance for increasing numbers of n. In 2005, Leskovec et al. analyzed the average distance in growing real–world networks [30,31]. They found that actually the diameter of growing graphs *shrinks* in some cases instead of growing (poly–)logarithmically with increasing n. They also present some new network models that describe this behavior. We refer the interested reader to their paper [30].

Another, even more important structural property of networks was found in 1999, the so–called *scale–freeness* of real–world networks.

4.2 Scale–Free Networks

Given a graph G let $P(k)$ denote the probability to choose a vertex with degree k uniformly at random from all vertices of the graph. A complex network is said to be scale–free when $P(k)$ scales with $k^{-\gamma}$ where γ is a positive constant [6]. This distribution is also said to follow a *power law*[3]. For real–world networks, γ is often between 2 and 3, see [38, Table 3.7] for an overview of n, m, and γ of around 30 different network types. The easiest way to detect a scale–free distribution is to compute $P(k)$ and to display it in dependence of k in a double logarithmic diagram. Since $\log P(k) \simeq -\gamma \log k$ the resulting plot shows a

[3] Note that power laws have different names like 'Lotka's law' or 'Zipf's law' and were investigated much earlier in other disciplines. E.g., 1926 Lotka observed that citations in academic literature might follow a power law [34].

straight line. Note however that if the available data does not span a large order of magnitudes it is actually hard to differentiate between power–laws and other possible distributions. Clauset et al. describe how to make more accurate estimates of the parameters governing a power–law distribution [17]. One such scale–free degree distribution in a real–world complex network was observed by the Faloutsos brothers in 1999 when they analyzed a sample of the weblink graph [20].

As is obvious by the construction of a random graph, its degree distribution follows a normal distribution (for large n). In comparison with this, a scale–free graph with the same number of vertices and edges as a corresponding random graph has much more low–degree vertices and also some vertices with a much higher degree than to be expected in a random graph. These high–degree vertices are also called *hubs*.

The first model that explained how such a scale–free degree distribution can emerge is the *preferential attachment* or *the–rich–get–richer* model by Barabási and Albert [6]: it is a dynamic network model where in each step i one new vertex v_i is added together with k incident edges. It starts with a small random graph of at least k vertices. Subsequently, each new vertex v_i chooses k vertices from the already existing ones, each with a probability that is proportional to its degree at this time point and creates an edge to it. More precisely, the probability that the newly introduced vertex v_i chooses vertex w is proportional to $deg(w)$ at that time point. Thus, if a vertex already has a large degree, it has a higher chance to get a new edge in each time step, a process which is called *preferential attachment*. This procedure produces a scale–free network in the limit of large n [1]. Note that the original model is not defined in every detail, especially the starting graph and the exact choice mechanism are not fully defined. As Bollobás and Riordan point out, different choices can lead to different results [10]. Their LCD model is defined rigorously and can thus be analyzed more easily [11]. For an overview on other scale–free producing network models see also [7].

Many real–world networks are reported to have scale–free degree distributions, the Internet [20], the weblink graph [6], or the web of human sexual contacts [32,33], to name just a few. Especially the findings on the sexual contact network have important consequences: first, it can explain why sexual diseases are so easily spread and second, it can also help to prevent the spreading, by finding especially active hubs and treat them medically. This follows from the fact that scale–free networks are most easily disconnected if only a small fraction of the high–degree vertices are removed as we will discuss in Section 4.3. The scale–free nature of email contact networks can also explain why some (computer) viruses stay nearly forever in the network: Pastor-Satorras and Vespignani discuss a model of virus spreading over a scale–free network and show that in this network there is no *epidemic threshold*, i.e., no minimal infection density to enable the infection of nearly the whole network [41]. Thus, in such a network, every virus can potentially infect the whole network.

4.3 Robustness of Random and Scale–Free Networks

One very interesting finding of Albert et al. is that different network structures show very different robustness against random failures and directed attacks against their structure [2]. They defined the *robustness* of a network as the average distance after a given percentage of the vertices were removed from the network. The removal is modeled in two ways: to model a random failure of, e.g., a server in the internet, any vertex is chosen uniformly at random to be removed from the network; to model a directed attack of some malicious adversary that knows the network structure, the vertex with highest degree is removed from the network. Albert et al. could show that in a random failure scenario the robustness of a scale–free network is much higher than that of a corresponding random graph. Furthermore, after removing more and more vertices, the random graph finally disconnects into many small connected components while most vertices in the scale–free network are still forming a big connected component. But for an attack scenario, the reverse is true: while the random graph stays connected and shows a rather low average distance, the scale–free network will decompose after just a few high–degree vertices are removed. This behavior is easily explained: in a random failure scenario, most of the removed networks in a scale–free network will have a very small degree since most vertices in a scale–free network have a small degree. In a random graph, almost all vertices have the same degree and the same importance for the connectedness of the graph. This property saves the network in the case of directed attacks. But the scale–free network will lose a high percentage of its edges very quickly if its high–degree vertices are removed which makes it very vulnerable to this kind of attack. This result, although very easily explained when it was discovered, is quite devastating since most of our communication and even some of our transportation networks, especially flight networks, are scale–free. Thus, Albert et al. showed how easily these networks might become disconnected. We will show in the following section that this problem can be alleviated by allowing the network to react to the attacks.

5 Network Design for Systems with Independent Agents

In many complex systems there is no central authority, they are decentrally organized. In these networks, it is, e.g., very hard for a single participant to understand whether a missing neighbor is missing due to a random failure or a directed attack since this requires a global overview. Nonetheless, because of the different robustness of random and scale–free network structures (s. Subsec. 4.3) it would be very convenient if a network could change its structure according to the situation it is in, i.e., to a random network in the case of attacks and to a scale–free network in the case of random failures. In the following section we will show that this can be achieved in a decentrally organized network. In Sec. 5.2 we will then show how carefully a network generating protocol has to be implemented in a decentrally organized system because a subtle change can make the difference between an efficient and an inefficient network evolution.

5.1 Adaptive Network Structures

In [50] we considered the question of whether a decentrally organized network can adapt its network structure to its environmental situation, i.e., a random failure scenario or an attack scenario, while the single participants are oblivious of this situation. The first thing to observe is that it is not necessary to have a really random network structure in the case of attacks, it suffices to have a network in which almost every vertex has the same degree. It is also not necessary to make a real scale–free network in which the degree distribution is described by $P(k) \simeq k^{-\gamma}$. It suffices if the degree distribution is sufficiently right–skewed, i.e., if most of the vertices have a low degree and some a very high degree. The second observation to be made is that in the case of attacks, the wanted change towards a more uniform degree distribution is essentially achieved by the nature of the attack itself: since it removes high–degree vertices it smoothes the degree distribution. A third observation is that in the model of Albert et al. the vertices are not allowed to react to the situation. Thus, in the scale–free network the attack on only a few high–degree vertices already removes a high percentages of the edges. We will now present a network re–generating protocol that is decentral and oblivious of the situation in which the network is in and achieves to adapt the network's structure to the one best suited for the situation. Consider the following model: In each time step remove one vertex x at random (random failure scenario) or remove the vertex with the highest degree (attack scenario). A vertex v notices if its neighbor x is missing. Vertex v will now build one edge to any of its neighbors w in distance 2 with probability 0.5. Let this set of neighbors in distance 2 be denoted by $N_2(v)$. The probability with which a vertex v chooses w is computed by the following generic formula:

$$p_i(v,w) = \frac{deg(w)^i}{\sum\limits_{w' \in N_2(v)} deg(w')^i}. \tag{6}$$

Thus, if $i = 0$ all second–hand neighbors have the same probability to be chosen, if $i = 1$ the process resembles a local preferential attachment. Algorithm 1 describes this re–generating behavior in pseudo code. In the following $A0$ will denote this algorithm where $p_i(v,w)$ is computed with $i = 0$ and $A1$ denotes the algorithm where i is set to 1.

Note that to be able to compare the robustness of the resulting networks it is necessary to keep the number of vertices and edges constant. Thus, the removed vertex x is allowed to re–enter the network, building edges at random to other vertices. To keep the number of edges (at least approximately) constant x will build half as many edges as it had before the removal[4]. Thus, since every of its former neighbors builds a new edge with probability 0.5 and itself builds another $deg(x)/2$ edges, the number of edges stays approximately constant.

[4] We tried many different variations of re–inserting the removed vertex. The general outcome did not seem to be influenced by the details of this procedure. It should be noted that the method sketched here neither introduces a skewed nor a very narrow degree distribution on its own.

Algorithm 1. Algorithm for rewiring a deleted edge to one of the second neighbors

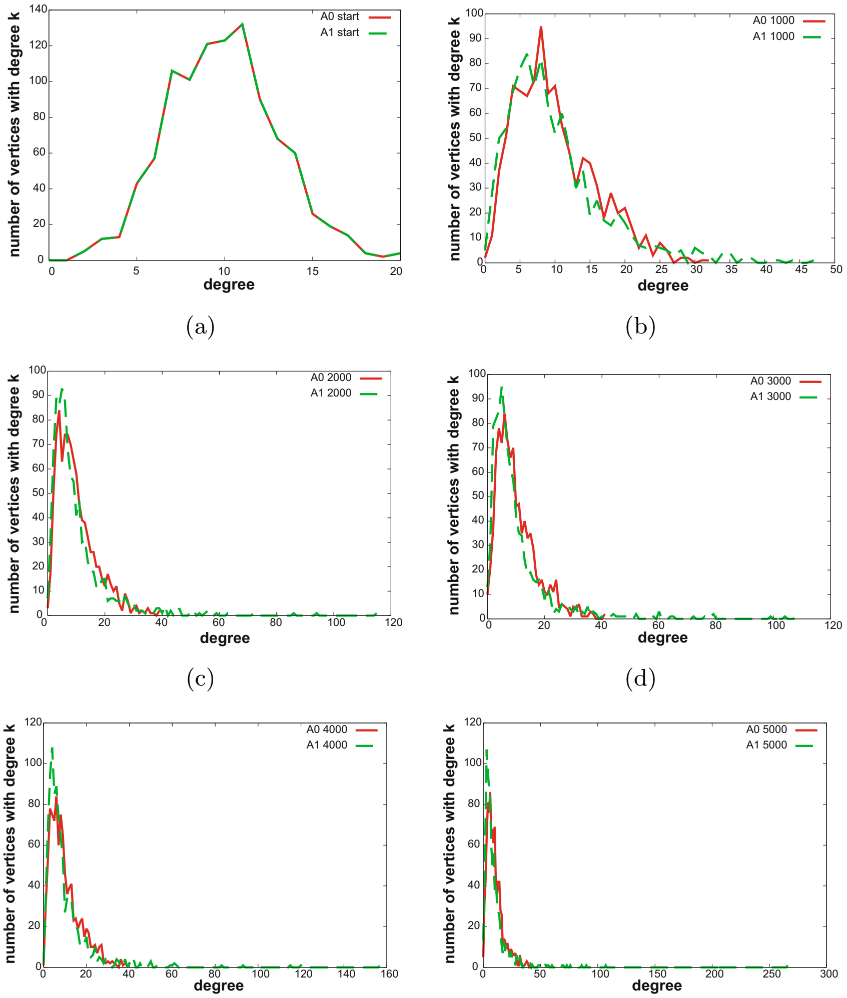
```
procedure NODE.REWIRE(NODE v)                                          ▷
    if (any neighbor of node v is deleted) then
        if (random.nextDouble() < 0.5) then
            target ← choose second neighbor $w$ with probability $P_i(v, w)$);
            create edge between node v and target;
        end if
    end if
end procedure
```

Consider now a random graph that suffers from random failures. In this scenario, we would like the graph to establish a right-skewed degree distribution. Fig. 3 shows that algorithm $A1$ is faster in building a right–skewed degree distribution than $A0$. This implies that also a local preferential attachment is enough to create something like a scale–free degree distribution.

Remember that a real scale–free network structure stabilizes the network against random failures, i.e., it keeps a low average distance even if a substantial part of the vertices are removed at random. It also makes the network more fragile against directed attacks. To measure this effect we use Albert et al.'s definition of robustness: the *robustness* of a graph as introduced by Albert et al. [2] is measured by the *average distance between all vertices* after a given percentage of nodes is removed from the graph (without any rewiring). In the following we will set this value to 5%. If robustness against attacks is measured, the removed vertices are the 5% vertices with highest degree, in the case of random failures the set is chosen uniformly at random. The robustness measures are denoted by $\mathcal{R}_A(G)$ for attack robustness and by $\mathcal{R}_{RF}(G)$ for random failure robustness. Note that this measure as introduced by Barabási and Albert is a bit unintuitive since a *higher* value denotes a *less* robust network and a *lower value* denotes a *more* robust network.

To analyze whether $A1$ creates a robust network against random failures and a fragile network with respect to attacks, we started with a random graph with $1,000$ vertices and $5,000$ edges. This network then experiences $20,000$ random failures. After $1,000$ failures each, the resulting graph is taken and $R_A(G)$ and $R_{RF}(G)$ are measured. After that, the next $1,000$ random failures are simulated together with the re–generating algorithms $A0$ and $A1$, respectively.

In a pure random graph with $1,000$ vertices and $5,000$ edges $R_A(G)$ is 3.4 and an $R_{RF}(G)$ is 3.3, i.e., as expected the increase in the average path length is very similar and only slightly higher in the attack scenario than in the random failure scenario. In a pure scale–free network with the same number of vertices and edges, R_A is higher than that in a random graph, namely 3.5. As expected, the robustness against random failures in a pure scale–free graph is much better than that of the random graph with $R_{RF}(G)$ being 3.0. Thus, we expect that after some random failures the robustness measures after applying $A0$ and $A1$ should match that of a pure scale–free graph, i.e., approach 3.5 for $R_A(G)$ and 3.0 for $R_{RF}(G)$ (s. Fig. 4).



(a)

(b)

(c)

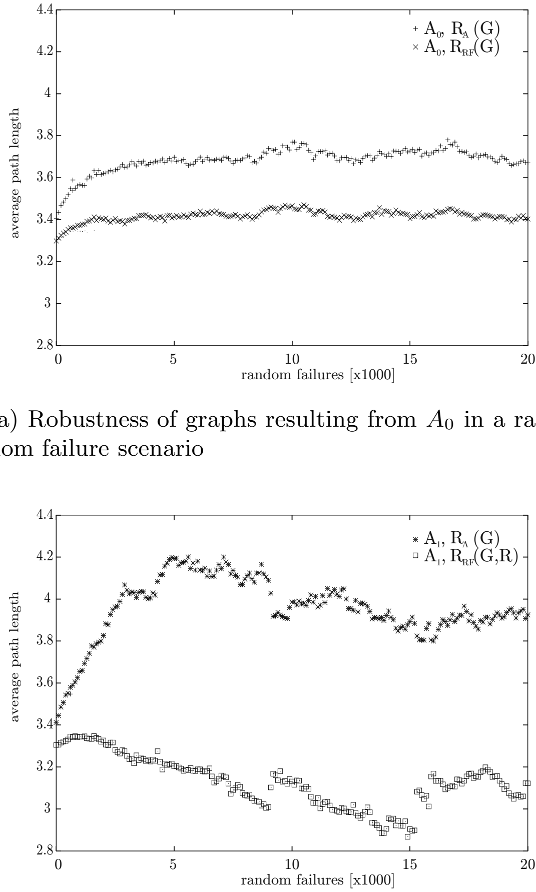
(d)

(e)

(f)

Fig. 3. Exemplary evolution of the degree distribution of one random graph after 5, 000 random failures, plotted after every 1000 deletions. (a) Since both runs start with the same random graph with $n = 1000$ and $m = 5,000$, the degree distribution is the same for both algorithms. (b)-(f) Each diagram compares the resulting degree distribution after applying algorithm $A0$ and $A1$. It is clear to see that $A1$ results in a degree distribution that is more right-skewed than the one created by $A0$. For example, in diagram f, the highest degree in the graph resulting from procedure $A1$ is around 250, that of $A0$ is around 50.

Algorithm $A0$ does not achieve this goal, even after 20, 000 random failures its robustness in both cases, random failures and attacks, is worse than that of a pure random graph or a pure scale–free graph (s. Table 1). Algorithm $A1$



(a) Robustness of graphs resulting from A_0 in a random failure scenario

(b) Robustness of graphs resulting from A_1 in a random failure scenario

Fig. 4. Evolution of $\mathcal{R}_A(G)$ and $\mathcal{R}_{FT}(G)$ in a long random failure scenario with 20,000 events and application of $A0$ and $A1$. Starting graph is a random graph with 1000 vertices and 5000 edges. (a) Algorithm $A0$ creates a graph that is less robust against attacks than a pure random graph: its average path length after removing 5% of the vertices is 3.6 compared to 3.4 in a pure random graph. It is also higher (less robust) than the value in a pure scale–free graph which has $\mathcal{R}_A(G) = 3.5$. The graph's robustness against random failures is worse than that of a pure random graph (3.4 vs. 3.3). (b) As expected, algorithm $A1$ is able to create a graph that is at least as robust against random failures as a comparable scale–free graph ($\mathcal{R}_A(G) \simeq 2.9 - 3$ compared to 3.0 of a pure scale–free graph). Accordingly, its robustness against attacks is even worse than a comparable scale–free graph ($\simeq$ 4 vs. 3.5), i.e., the resulting graph's short paths are strongly depending on the high degree vertices. Note that jumps in the curves are caused by deletion of a high degree vertex by chance.

Table 1. Comparison of the robustness of pure random graphs and pure scale–free networks with the networks that result after applying 20,000 random failures and either regenerating algorithm $A0$ or $A1$. It is clear to see that the network resulting with algorithm $A0$ performs worse than both the pure random and the pure scale–free graph, while the graph resulting from $A1$ comes near to the robustness of a pure scale–free graph in the case of attacks. On the other hand, it is even more fragile in the case of directed attacks. This is due to the localized structure of the network regenerating algorithm.

	Random Graph	Scale–Free Graph	$A0$ after $20,000$ steps	$A1$ after $20,000$ steps
$R_A(G)$	3.4	3.5	3.6	3.9
$R_{RF}(G)$	3.3	3.0	3.4	3.1

produces a network that is nearly as robust against random failures as the pure scale–free graph which is astonishing since it only uses local information. On the other hand it produces networks that are even more fragile with respect to attacks than pure scale–free graphs. But, as we could show, this is not a huge problem since the nature of the attack will very quickly move the network's degree distribution back to a narrow and thus robust distribution.

In summary, this model shows that the fragility of scale–free networks can be alleviated if the network is allowed to react to missing vertices. Of course this is not always possible on a short time–scale, e.g., in street networks. But for example in flight networks, a redirection of airplanes is only a minor problem. Also in P2P–networks a new edge can be built fast and without high costs.

As we have shown in the previous paragraphs, many relevant technical networks show specific network structures that influence the behavior of processes running on top of them. To analyze the behavior of a new communication protocol or the runtime of a graph algorithm on real–world networks, it is very important to analyze the network's structure and to model it as closely as possible. But sometimes computer scientists face the opposite perspective, namely to design a network such that its structure supports the processes on top of it. This is already a difficult task if the network is static and centrally organized, but nowadays it becomes more and more important to design network generating rules between independent and maybe even mobile agents, like in sensor networks, peer–to–peer networks [28], or robot swarms. In the following section we will show how difficult this design task can be, even in a toy example.

5.2 The Sensitivity of Network Generating Algorithms in Decentrally Organized Systems

Social systems are among the most complex systems to explore because their global structure depends on the individual decisions of the humans that constitute them. We can assume that humans will interact with each other when both profit from this interaction in a way, i.e., we will assume that the human agents are selfish. In internet–based communication networks, foremost P2P–networks, these decisions are mediated by the software that manages the communication over the internet. A P2P–network can be seen as an overlay network of the internet, i.e., every user has a *buddy list* of other participants with which she can communicate directly. If a query, e.g., for a file, cannot be answered by one of her neighbors, the query will be redirected to (all or a choice of) the neighbors of her neighbors and so on. The complex network of this system thus represents each participant p_i as vertex v_i, and v_i is connected to those vertices that represent the participants on the buddy list of p_i. As already sketched above, some network structures are more favourable than others. For example, it might be wanted that a P2P–network has a small diameter. This sounds like a network property that has to be managed centrally. On the other hand, we know that if every participant had only one random edge the diameter would already scale poly–logarithmically. But why should a random edge be valuable for a single participant? In the example of P2P–networks, especially file–sharing networks, it is much more valuable to be connected to those participants that have a similar interest than to any random participant which may never have any interesting file to share. Thus, the question is: what kind of control can be excerted to guide the decisions of each single user such that an overall favourable network structure arises? Especially in software mediated communication networks, the designer can indeed excert some control, namely by the information about the current network structure the software feeds back to the users. Consider the following toy example: The initial starting configuration is a tree. Each participant v is told its *eccentricity* $ecc(v)$, i.e., its maximal distance to any other participant in the network:

$$ecc(v) := \max_{w \in V} d(v, w). \tag{7}$$

Given this information, a vertex is *satisfied* if this eccentricity does not exceed some constant k. In each step, one participant v is chosen at random. If its eccentricity is larger than k, it tries to improve its position in the network: to do so, let $N_2(v)$ denote the vertices in distance 2 from v that are no leaves. Now, v chooses one of these vertices z uniformly at random. Let w denote the mediating neighbor of v and z (s. Fig. 5).

The edge (v, w) is then temporarily removed and the edge (v, z) is temporarily built. If the eccentricity of v does not increase by this new edge, v will keep the edge and otherwise the original edge (v, w) is re–established. It can be shown that this procedure will eventually form a tree with diameter k, i.e., a tree in which all participants are satisfied [49, Lemma 6.1]. Of course, it is important how fast this globally satisfying structure will be achieved. Let now k be equal to 2, i.e., the globally satisfying network structure is a star. Unfortunately, the following theorem shows that this will take expectedly exponential time:

Theorem 1. *[27,49, Theorem 6.1]*
Starting with a tree with diameter larger than 2*, the expected runtime to generate the star with the above sketched mechanism is bounded from below by* $\Omega(2^n)$.

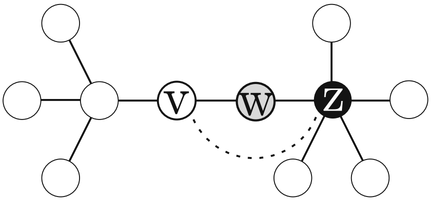


Fig. 5. One vertex v is chosen at random in every time step. If its eccentricity is greater than k it will try to connect to a non-leaf vertex in distance 2 (black vertex). Let z be the second neighbor chosen and w be the vertex connecting both. Then edge (v, w) will be replaced by edge (v, z) if the eccentricity of v does not increase due to this process.

This situation can be greatly improved by feeding back the *closeness* $close(v)$ of each participant instead of its eccentricity:

$$close(v) := \sum_{w \in V} d(v, w). \tag{8}$$

By assuming that each participant will only accept a new edge if its closeness is strictly decreased, it can first be shown that a globally satisfying network is finally achieved [27,49, Lemma 6.2]. Furthermore, the expected time until this happens is polynomial:

Theorem 2. *[27,49, Theorem 6.2]*
If the closeness is fed back to each participant, the expected runtime until an overall satisfying network structure is generated is bounded by $O(n^5)$.

By a more involved analysis, Jansen and Theile [24] improved the upper bound to $O(n^{5/2})$ [Th. 4] and could show that the runtime is bounded from below by $\Omega(n \log n)$ [Th. 3]. Of course, this toy model of a complex system generated by independent, selfish agents is not easily applied to realistic situations, in which we have to deal with graphs instead of trees, in which agents change the network at the same time (asynchronous behaviour), and in which it is also unlikely to be able to compute the closeness or eccentricity. But it proves the following two points:

1. Even in a system comprised of independent agents, a designer can excert some control over the network generating process by feeding back a well–designed subset of structural information.
2. The choice of which information to feed back must be made very carefully since it can make the difference between an exponential and a polynomial runtime until a satisfying network structure is achieved.

6 Summary

The last years have shown that real–world networks have a distinct structure that is responsible for the behavior of various processes that take place on them.

We have cites some papers that indicate that these structures also influence the behavior of technical networks, especially the internet or peer–to–peer networks. It could additionally be shown empirically that a small–world structure is likely to change the runtime of algorithms solving NP-hard problems on these graphs [45]. On the other hand, the complexity of, e.g., coloring a graph is unchanged on scale–free networks [21]. In any case, we think that further research should be directed to find and analyze those real–world structures that can be used to design more efficient algorithms or that make a problem even harder to solve in practice. As cited above, scale-free networks show a distinct behavior in computer simulations for, e.g., virus spreading, and thus simulations of similar processes should be based on an appropriately chosen network model that captures the essential structures of the according real–world network. Last but not least, computer scientists are more and more asked to design communication networks between humans and large swarms of mobile and independent devices. We have shown that it is indeed possible to control the network generating process even in a decentrally system of independent agents to achieve various, globally satisfying network structures, but we have also shown that a careful network generating protocol design is needed to do so. In summary, complex network science is important for computer scientists, as well in algorithm engineering as in the design of technical networks.

7 Further Reading

As sketched in Sec. 1, the field of complex network science can be divided into three areas: analysis, models, and processes on complex networks. At the moment there is no textbook that comprises all of these fields. We will thus refer to some review articles or books in each of the fields.

Network analysis was done long before the 1990s, especially in the social sciences. The textbook by Wasserman and Faust [46] is a classic book in that realm. The book edited by Brandes and Erlebach covers most of these older and some of the newer results and moreover makes the effort to present them in a well–defined, formal framework [13]. The classic book on the analysis of random graph models is of course the one by Bollobás [9]. A new one that takes other, more recent random graph models into account is that by Chung and Lu [15].

Also the later results that are predominantly published by physicists have not yet found their way in one, comprehensive textbook. The early book by Dorogovtsev and Mendes covers almost only scale–free networks [18]. It is helpful since it explains some of the physics models used in this approach quite nicely. Watts has published his Ph.D. thesis which covers small–world network models [47]. Albert and Barabási have published a long review article (based on Albert's thesis) which is very readable and covers mostly scale–free network models and their behavior [1]. An interesting overview of applied complex network science is given in a handbook edited by Bornholdt and Schuster [12]. A very good collection of original papers, partitioned into five chapters alongside with a comprehensive introduction to each, was edited by Barabási, Watts, and Newman

[38]. Alon has published a book that covers his findings on patterns in biological networks [3].

To our knowledge there is no comprehensive article or book that covers the behavior of processes on networks in dependency of their structure. Thus, we refer the reader to the last chapter of the article collection by Barabási, Watts, and Newman where at least some of this work can be found [38].

References

1. Albert, R., Barabási, A.-L.: Statistical mechanics of complex networks. Review of Modern Physics 74, 47–97 (2002)
2. Albert, R., Jeong, H., Barabási, A.-L.: Error and attack tolerance of complex networks. Nature 406, 378–382 (2000)
3. Alon, U.: An Introduction to Systems Biology: Design Principles of Biological Circuits. Chapman & Hall/CRC, Boca Raton (2006)
4. Andersen, R., Chung, F., Lu, L.: Analyzing the small world phenomenon using a hybrid model with local network flow. In: Leonardi, S. (ed.) WAW 2004. LNCS, vol. 3243, pp. 19–30. Springer, Heidelberg (2004)
5. Barabási, A.-L.: Linked - The New Science of Network. Perseus, Cambridge (2002)
6. Barabási, A.-L., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509–512 (1999)
7. Baumann, N., Stiller, S.: Network Models. In: Brandes, U., Erlebach, T. (eds.) Network Analysis. LNCS, vol. 3418, pp. 341–372. Springer, Heidelberg (2005)
8. Berners-Lee, T., Hall, W., Hendler, J.A., O'Hara, K., Shadbolt, N., Weitzner, D.J.: A framework for web science. Foundations and Trends in Web Science 1(1), 1–130 (2006)
9. Bollobás, B.: Random Graphs, 2nd edn. Cambridge Studies in Advanced Mathematics, vol. 73. Cambridge University Press, London (2001)
10. Bollobás, B., Riordan, O.M.: Mathematical results on scale-free random graphs. In: Handbook of Graphs and Networks, pp. 1–34. Springer, Heidelberg (2003)
11. Bollobás, B., Riordan, O.M.: The diameter of a scale–free random graph. Combinatorica 24(1), 5–34 (2004)
12. Bornholdt, S., Schuster, H.G. (eds.): Handbook of Graphs and Networks. Wiley-VCH, Weinheim (2003)
13. Brandes, U., Erlebach, T. (eds.): Network Analysis - Methodological Foundations. Springer, Heidelberg (2005)
14. Chung, F., Lu, L.: The small world phenomenon in hybrid power law graphs. In: Ben-Naim, E., Frauenfelder, H., Toroczkai, Z. (eds.) Complex Networks, pp. 91–106 (2004)
15. Chung, F., Lu, L.: Complex Graphs and Networks. American Mathematical Society, Providence (2006)
16. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Physical Review E 70, 066111 (2004)
17. Clauset, A., Shalizi, C.R., Newman, M.E.J.: Power–law distributions in empirical data. ArXiv (June 2007)
18. Dorogovtsev, S.N., Mendes, J.F.F.: Evolution of Networks. Oxford University Press, Oxford (2003)
19. Erdős, P., Rényi, A.: On random graphs. Publicationes Mathematicae 6, 290–297 (1959)

20. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. Computer Communications Review 29, 251–262 (1999)
21. Ferrante, A., Pandurangan, G., Park, K.: On the hardness of optimization in power-law graphs. Theor. Comput. Sci. 393(1-3), 220–230 (2008)
22. Gilbert, E.N.: Random graphs. Anual Math. Statist. 30, 1141–1144 (1959)
23. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proceedings of the National Academy of Sciences 99, 7821–7826 (2002)
24. Jansen, T., Theile, M.: Stability in the self-organized evolution of networks. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 931–938 (2007)
25. Kleinberg, J.: Navigation in a small world. Nature 406, 845 (2000)
26. Kleinberg, J.: The small-world phenomenon: An algorithmic perspective. In: Proceedings of the 32nd ACM Symposium on Theory of Computing, pp. 163–170 (2000)
27. Lehmann, K.A., Kaufmann, M.: Evolutionary algorithms for the self-organized evolution of networks. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005), pp. 563–570 (2005)
28. Lehmann, K.A., Kaufmann, M.: Random Graphs, Small Worlds and Scale-Free Networks. In: Steinmetz, R., Wehrle, K. (eds.) Peer-to-Peer Systems and Applications. LNCS, vol. 3485, pp. 57–76. Springer, Heidelberg (2005)
29. Lehmann, K.A., Post, H.D., Kaufmann, M.: Hybrid graphs as a framework for the small-world effect. Physical Review E 73, 056108 (2006)
30. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graphs over time: Densification laws, shrinking diameters, and possible explanations. In: Proceedings of the 11th ACM SIGKDD (2005)
31. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graph evolution: Densification and shrinking diameters. ACM Transactions on Knowledge Discovery from Data (TKDD) 1(1, 2) (2007)
32. Liljeros, F.: Sexual networks in contemporary western societies. Physica A 338, 238–245 (2004)
33. Liljeros, F., Edling, C.R., Amaral, L.A.N., Stanley, H.E., Åberg, Y.: The web of human sexual contacts. . Nature 411, 907–908 (2001)
34. Lotka, A.: The frequency distribution of scientific productivity. Journal of the Washington Academy of Sciences 16, 317–323 (1926)
35. Milo, R., Itzkovitz, S., Kashtan, N., Levitt, R., Shen-Orr, S., Ayzenshtat, I., Sheffer, M., Alon, U.: Superfamilies of evolved and designed networks. Science 303, 1538–1542 (2004)
36. Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network motifs: Simple building blocks of complex networks. Science 298, 824–827 (2002)
37. Mark, E.J.: The structure of scientific collaboration networks. Proceedings of the National Academy of Sciences 98(2), 404–409 (2001)
38. Newman, M.E.J., Barabási, A.-L., Watts, D.J. (eds.): The Structure and Dynamics of Networks. Princeton University Press, Princeton (2006)
39. Nunkesser, M., Sawitzki, D.: Blockmodels. In: Brandes, U., Erlebach, T. (eds.) Network Analysis. LNCS, vol. 3418, pp. 253–292. Springer, Heidelberg (2005)
40. Palla, G., Derényi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. Nature 435, 814–818 (2005)
41. Pastor-Satorras, R., Vespignani, A.: Epidemic spreading in scale-free networks. Physical Review Letters 86(4), 3200–3203 (2001)
42. Robins, G., Pattison, P., Kalish, Y., Lusher, D.: An introduction to exponential random graph (p^*) models for social networks. Social Networks 29, 173–191 (2007)

43. Robins, G., Snijder, T., Wang, P., Handcock, M., Pattison, P.: Recent developments in exponential random graph (p^*) models for social networks. Social Networks 29, 192–215 (2007)
44. Song, C., Havlin, S., Makse, H.A.: Origins of fractality in the growth of complex networks. Nature 2, 275–281 (2006)
45. Walsh, T.: Search in a small world. In: Proceedings of the IJCAI 1999 (1999)
46. Wasserman, S., Faust, K.: Social Network Analysis - Methods and Applications, revised, reprinted edn. Cambridge University Press, Cambridge (1999)
47. Watts, D.J.: Small Worlds- The Dynamics of Networks between Order and Randomness. Princeton Studies in Complexity. Princeton University Press, Princeton (1999)
48. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature 393, 440–442 (1998)
49. Zweig, K.A.: On Local Behavior and Global Structures in the Evolution of Complex Networks. Ph.D thesis, University of Tübingen, Wilhelm-Schickard-Institut für Informatik (2007)
50. Zweig, K.A., Zimmermann, K.: Wanderer between the worlds – self-organized network stability in attack and random failure scenarios. In: Proceedings of the 2nd IEEE SASO (2008)

Algorithms and Simulation Methods for Topology-Aware Sensor Networks

Alexander Kröller[1], Dennis Pfisterer[2], Sándor P. Fekete[1], and Stefan Fischer[2]

[1] Algorithms Group
Braunschweig Institute of Technology
{kroeller,fekete}@tu-bs.de
http://www.ibr.cs.tu-bs.de/alg/

[2] Institute of Telematics
University of Lübeck, Germany
{pfisterer,fischer}@itm.uni-luebeck.de
http://www.itm.uni-luebeck.de

Abstract. This chapter presents a number of different aspects related to a particular kind of large and complex networks: A Wireless Sensor Network (WSN) consists of a large number of nodes that individually have limited computing power and information; their interaction is strictly local, but their task is to build global structures and pursue global objectives.

Dealing with WSNs requires a mixture of theory and practice, i.e., a combination of algorithmic foundations with simulations and experiments that has been the subject of our project SwarmNet. In the first part, we describe a number of fundamental algorithmic issues: boundary recognition without node coordinates, clustering, routing, and energy-constrained flows. The second part deals with the simulation of large-scale WSNs; we describe the most important challenges and how they can be tackled with our network simulator Shawn.

1 Introduction

Our modern world has grown immensely complex, much beyond what any group of human experts could have designed. A major part in this is played by *decentralized networks*, which form a robust yet dynamic mechanism that is able to handle a large variety of challenges. In the following we give a brief overview of a particular type of decentralized network, as well as the corresponding algorithmic and simulation challenges.

1.1 Decentralized Networks

Traditional computing systems are based on a centralized algorithmic paradigm: Data is gathered, processed, and the result administered by one central authority. On the other hand, in today's communication world, wireless networks such as the mobile phone network GSM in the wide area and WiFi in the local area

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 380–400, 2009.

range have become ubiquitous. So far, most applications using these networks rely on a thoroughly managed infrastructure, e.g., base stations in GSM or access points in WiFi. Many research activities, however, already go one step further and make use of the fact that more and more mobile devices with radio communication capabilities are available. These devices are not necessarily bound to communication infrastructures, but can instead create a spontaneous network, a so-called ad-hoc network, in order to allow communication among all participating nodes (and not necessarily to the outside). In its sophisticated form, some of the nodes in an ad-hoc network act as relay stations and transport data from a source to a destination that is not in direct radio range (multihop ad-hoc network).

Based on the ad-hoc kind of network setup, another new kind of network is currently in the focus of many research activities: the *sensor network.* Its purpose is the recording, transport and interpretation of phenomena of the environment, such as measuring temperatures, monitoring areas, etc. Many people claim that this new technology is so substantially different from today's networks that from the scientific point of view a paradigm shift for design, implementation, deployment and management of such networks and their nodes is necessary. A number of facts support this view: Sensor networks usually consist of a large number of nodes, while their communication mode is usually ad hoc without infrastructure. It may be necessary that after potentially uncontrolled deployment (e.g., throwing nodes out of a plane), nodes have to organize themselves into a network without the help of an administrator. During the network's lifetime, nodes might move, run out of energy or may even be destroyed. Sensor nodes have very limited computing and storage capacities; on the other hand, algorithms have to be devised with limited energy supply in mind. Finally, sensor networks usually provide enormous amounts of data. In order to get meaningful results from their operation, powerful data aggregation and interpretation facilities have to be created.

1.2 Algorithmic Challenges

From an algorithmic point of view, the characteristics of a sensor network require working under a paradigm that is different from classical models of computation: absence of a central control unit, limited capabilities of nodes, and limited communication between nodes require developing new algorithmic ideas that combine methods of distributed computing and network protocols with traditional centralized network algorithms. In other words: How can we use a limited amount of strictly local information in order to achieve distributed knowledge of global network properties?

This task is much simpler if the exact location of each node is known. Computing node coordinates has received a considerable amount of attention. Unfortunately, computing exact coordinates requires the use of special location hardware like GPS, or alternatively, scanning devices, imposing physical demands on size and structure of sensor nodes. As we demonstrated in our paper [1], current methods for computing coordinates based on anchor points and distance estimates encounter serious difficulties in the presence of even small inaccuracies, which are unavoidable in practice.

Our work shows that even in the absence of coordinates, there is some underlying geometric information that can be exploited. As we discuss in Section 2, boundary recognition without node coordinates, clustering, routing, and energy-constrained flows allow algorithmic solutions.

1.3 Simulation Challenges

To acquire a deeper understanding of these networks, three fundamentally different approaches exist: Analytical methods, computer simulation, and physical experiments. Designing algorithms for sensor networks can be inherently complex. Many aspects such as energy efficiency, limited resources, decentralized collaboration, fault tolerance, and the study of the global behavior emerging from local interactions have to be tackled.

In principle, experimenting with actual sensor networks would be a good way of demonstrating that a system is able to achieve certain objectives, even under real-world conditions. However, this approach poses a number of practical difficulties. First of all, it is difficult to operate and debug such systems. This may have contributed to the fact that only very few of these networks have yet been deployed [2, 3, 4]. Real-world systems typically consist of roughly a few dozen sensor nodes, whereas future scenarios anticipate networks of several thousands to millions of nodes [5, 6]. Using intricate tools for simulating a multitude of parameters, it may be possible to increase the real-world numbers by a couple of orders of magnitude. However, the difficulty of pursuing this approach obfuscates and misses another, much more crucial issue: The vision of an efficient, decentralized and self-organizing network cannot be achieved by simply putting together a large enough number of sensor nodes. Instead, coming up with the right functional structure is the grand scientific challenge for realizing the vision of sensor networks. Understanding and designing these structures poses a great number of algorithmic tasks, one level above the technical details of individual nodes. As this understanding progresses, new requirements may emerge for the capabilities of individual nodes; moreover, it is to be expected that the technical process and progress of miniaturization may impose new parameters and properties for a micro-simulation.

In Section 3 we discuss a powerful alternative: Our simulator Shawn focuses on the algorithmic effects of interaction within the network, rather than trying to simulate the physical basics. This allows a large-scale simulation of algorithmic approaches to large and complex networks.

2 Topology-Aware Algorithms

An important issue in WSNs is *Localization* or *Positioning*, i.e., letting the nodes know where they are. This essentially adds geometry to what would otherwise be conceivable only as a graph. It is often assumed that location knowledge can only be established by providing coordinates in 2- or 3-dimensional space to the nodes. Unfortunately, it is not generally possible to equip all nodes with positioning devices such as GPS receivers, due to size and cost constraints, and, most importantly, because GPS receivers only function in environments that allow line-of-sight connections to satellites.

Practically all relevant localization scenarios boil down to the following setting: Given some (or even zero) nodes that know their position, and a graph following geometric constraints, find consistent positions for the other nodes. Sometimes, inter-node distance or angles are given. It is well known that these problems are NP-hard if there is any inaccuracy in the data [7]. In a prestudy [1] we confirmed that available heuristics produce unusable results also in realistic scenarios.

This motivated us to seek location knowledge that does not consist of coordinates, but topological and geometric structures instead.

2.1 Boundary Recognition

One topological problem is *Boundary Recognition*: Given a sensor network, find the boundary of the covered area. There is no formal definition of the problem, as there are many different meaningful definitions for the boundary. We are especially interested in networks without available position information, i.e., we are given a graph defined by a geometric embedding, but the embedding itself is unknown. It is easy to construct a graph that can be defined via different embeddings, each of which has a different boundary.

We consider the following setup: The network populates a geometric region $A \subseteq \mathbb{R}^2$, i.e., no "large" piece of A is unoccupied. Now we identify nodes that are located close to the region's boundary ∂A. We assume that the nodes follow the *d-Quasi Unit Disk Graph* model [8, Chapter 2], where nodes can always communicate when their distance is at most $d < 1$, can never communicate when their distance is more than 1, and nothing is assumed for distances in between. This model is a more realistic generalization of Unit Disk Graphs (which arise as the special case $d = 1$), as it allows for non-circular communication ranges.

Randomized Settings: A straightforward approach is the following: Assume the nodes are distributed on A following a uniform distribution. Nodes close to ∂A will have fewer neighbors than the average, as parts of their communication range lies outside of A. We developed an algorithm that detects nodes close to ∂A with high probability, provided sufficient density [9]. The algorithm works well in very high densities (over 100 neighbors on average), which are unlikely to appear in practice. We improved the algorithm with a set of heuristics in the same spirit [10], exploiting measures from social network analysis that are sensitive to how close a node is to the boundary. In experiments we confirmed that they work sufficiently in networks with as few as 20–30 neighbors.

Deterministic Settings: The major issue with randomized approaches such as the previous ones are that they assume a specific distribution of the nodes.

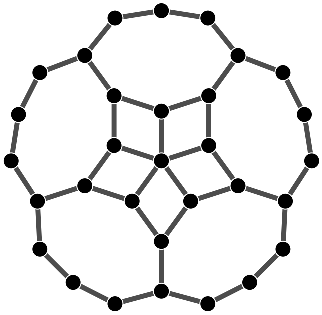


Fig. 1. A 5-flower

To overcome this, we proposed a deterministic algorithm that sidesteps all difficulties. Consider the graph shown in Figure 1, which is an example of a class of graphs called *flowers*. Flowers are beautiful because of the following fact [11]:

Theorem 1. *In every embedding as a d-Quasi Unit Disk Graph for $\sqrt{2}/2 < d \leqslant 1$, the outer cycle of a flower describes a polygon that contains the flower's central node on the inside.*

Proof sketch. Any embedding will have a number of properties: First, taking a cycle in the graph and all cycle neighbors produces a cut that cannot be crossed by any edge in the graph. This property vanishes when $d \leqslant \sqrt{2}/2$, which is the reason why d is constrained in the theorem. Consider a connected set U in the graph and such a cycle C with neighbors $N(C)$, where $U \cap (C \cup N(C)) = \emptyset$. It is obvious that either all of U is located outside of the polygon defined by C, or all of U is on the inside. At the same time, a set of independent nodes $I \subseteq U$ requires a certain area for the embedding, as the nodes cannot be placed close to each other. So, if $|I|$ is larger than a specific threshold defined by $|C|$, it can be concluded that U is located outside of the polygon. This is independent of the actual embedding. By applying this argument to cycles connecting the central node of a flower with one of the outer paths, one can prove the claim. □

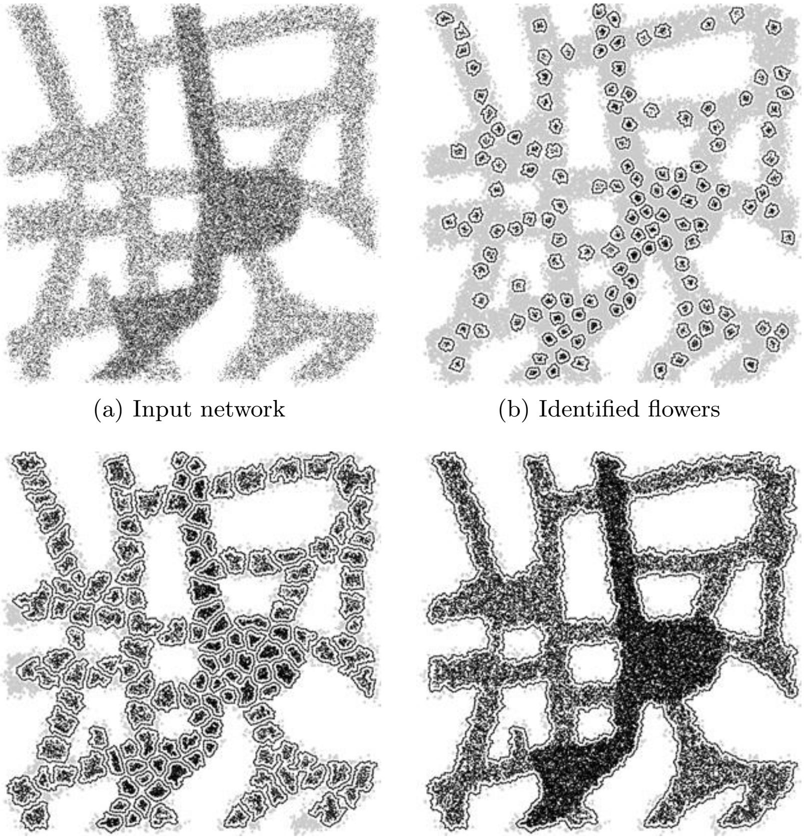
Theorem 1 provides a new way to detect boundaries: While a flower does not indicate anything about the network boundary, it does identify a subnetwork, and for it a meaningful and provably correct boundary.

We extend this idea to a global boundary recognition algorithm. We start by identifying one or more flowers, establishing a local "explored" region. From there, we successively enlarge the region by augmenting the flowers with small appendices. These also have the property that there is a cycle that provably surrounds some nodes. Thereby the algorithm extends the explored area until no further augmentation is possible. In the end, the algorithm produces one or more boundary cycles (if there are holes) and a set of "explored" nodes.

Our algorithm has two important properties that make it stand out:

- It is able to recognize the region for which it can identify the boundary; if the network has parts of very low density, the algorithm will only fail there.
- The boundary is described by a set of connected cycles instead of as a unordered set of nodes. This links the discrete representation to the topological shape of the explored region's boundary.

Figure 2 visualizes our algorithm when applied to a network in an urban scenario. Figure 2(a) shows the network, consisting of 50000 nodes with an average neighborhood size of 20 in the lighter and 30 in the darker regions. The following three pictures show the identified flowers, an intermediate state where explored regions are growing and merging, and finally the resulting boundary and the explored area.



(a) Input network

(b) Identified flowers

(c) Growing the explored region

(d) Final boundary

Fig. 2. Stages of the boundary recognition algorithm

2.2 Clustering

In the following we show how to utilize the detected boundary to provide location knowledge. Our goal is to segment $A \subseteq \mathbb{R}^2$ into regions $A_1, A_2, \ldots, A_k$. We then construct a cluster graph whose nodes refer to the region. There is an edge whenever the corresponding regions meet. Now each sensor node stores the cluster graph, which is of much smaller size than the sensor network, as well as the information to which cluster it belongs. If the clusters are constructed carefully, the resulting location knowledge accurately describes the network topology and can be applied in a number of applications.

Our clustering scheme is based on the *medial axis* $\mathbf{MA}(A)$ of A, which is the set of all centers of maximal circles contained in A. See Figure 3 for an example. The medial axis is a finite geometric graph under very mild assumptions on A [12]. Furthermore, it describes the topology of A accurately [13].

We construct two types of clusters:

- For every vertex x of $\mathbf{MA}(A)$, we construct a *vertex cluster*. Let $C(x)$ be the set of all contact points of x, that is $C(x) = S \cap \partial A$, where $S \subseteq A$ is a maximal circle centered at x. The set conv$C(x)$ becomes one cluster. See Figure 3 for a visualization.
- After all vertex clusters $A_1, \ldots, A_{k'}$ are constructed, we consider the connected components of $A \backslash (A_1 \cup \ldots \cup A_{k'})$. For each component B, we construct a *tunnel cluster* from its closure $\overline{B}$.

We can prove that our clusters have certain beneficial shapes [14]. Essentially, when thinking about A as the streets of a city, i.e., a highly complex topological shapes, vertex clusters take the role of intersections, and tunnel clusters reflect the streets. This makes the cluster graph equivalent to the street map of the city.

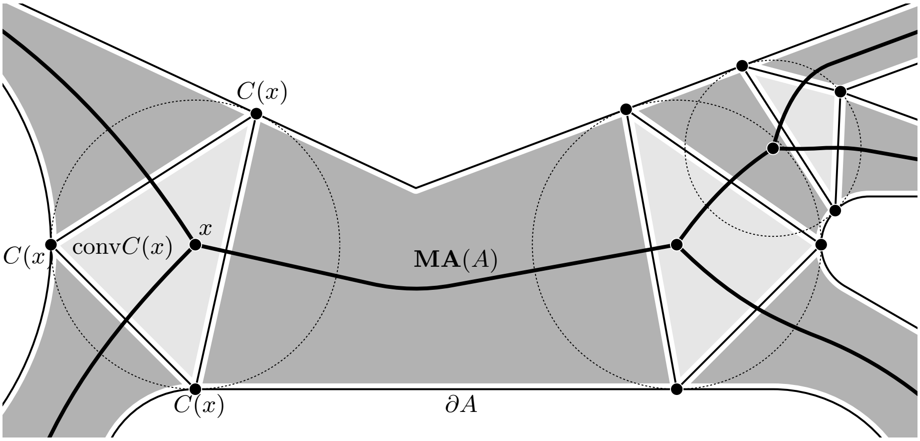


Fig. 3. Medial axis and vertex clusters for network region A

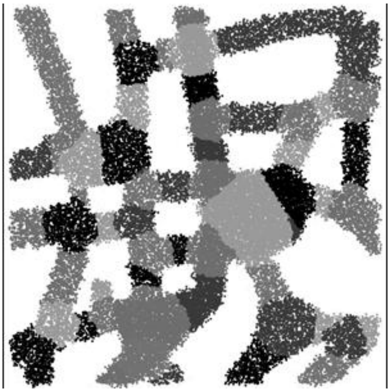


Fig. 4. Result of the clustering algorithm

Theorem 2. *The clustering adheres to the following properties:*

- *The clusters have pairwise disjoint interiors.*
- *Vertex clusters are convex.*
- *The number of clusters is finite.*
- *The boundary of a vertex cluster* conv$C(x)$ *is a finite alternating sequence of*
  1. *contact components of x and*
  2. *chords of the maximal circle around x.*
- *Let T be a tunnel cluster. Then one of the following holds:*
  1. *Either $T = A$, or*
  2. *∂T consists of one exit of some vertex cluster and one continuous piece of a boundary curve, or*
  3. *∂T consists of two vertex cluster exits (possibly two exits of the same cluster) and two continuous pieces of boundary curves.*

To apply the scheme in a discrete WSN, the definition has to be translated from geometric to network terms. This is easily done by using graph distances ("hop counts") as a replacement for Euclidean distances. The only problem lies in the definition of vertex clusters, where conv$C(x)$ has to be computed without available coordinates. Fortunately, such a cluster has a known shape. Our algorithm first constructs a cycle O representing ∂conv$C(x)$, consisting of shortest paths and boundary pieces. It then marks the interior by taking all nodes in the medial axis ball around x, and removes all those that can reach the outside without crossing O.

A sample output of our algorithm is depicted in Figure 4. It is clearly visible how some clusters exactly describe intersections, while others follow the streets.

2.3 Low-Overhead Routing

A straightforward application of a clustering scheme is message routing. To send a message to a node that cannot be reached directly, it is necessary to construct

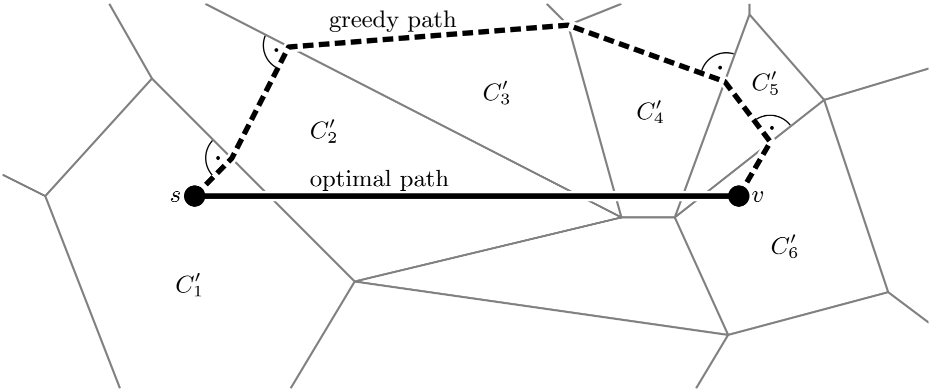


Fig. 5. Optimal vs. greedy path

a routing path through the network. A simple approach is that every node stores a successor for every possible destination, resulting in messages travelling along shortest paths. Unfortunately, such an approach requires a large amount of data to be stored in every node, if the number of possible destinations is unbounded.

Consider the case of an arbitrary node v communicating with a sink s of a WSN, that is a base station collecting data: If there is just a limited number of sinks, it is easy to send messages from v to s over a shortest path P_{vs}, using the aforementioned approach. However, routing responses (i.e., messages from s to v) is not as easy. We do not want the nodes on P_{vs} to store anything about messages they pass, because there could possibly be many simultaneous communications of nodes with the sink. Instead, we want to collect path information in the first packet from v to s, and use that for routing the responses. Simply collecting all nodes of P_{vs} in the packet is infeasible due to packet size limitations.

Our algorithm exploits a clustering. Assume the Euclidean plane to be partitioned into a finite set of convex clusters $C_1, \dots, C_k \subseteq \mathbb{R}$. Let $v, s \in \mathbb{R}^2$. The optimal, i.e., shortest path from s to v is a straight line segment. The path visits one or more clusters, let these be $C'_1, C'_2, \dots, C'_\ell$, in this order. Now we construct another v-s-path, called *greedy path*, as follows: Start at $s \in C'_1$. Connect s to the closest point in C'_2 using a straight line. Repeat connecting to the closest point in $C'_3, C'_4, \dots,$ until the path enters C'_ℓ. Finally connect to v directly. See Figure 5.

We were able to prove that the greedy path is an approximation [15, 16]:

Theorem 3. *The greedy path is a c-approximation to the optimal path, where* $\pi + 1 \leqslant c \leqslant 5$.

This directly translates into an efficient routing scheme: On the route from v to s, collect the visited clusters in the packet. Every node stores a successor on a shortest path for every neighboring cluster. In the cluster of v, either employ a recursive application of the scheme or temporarily store shortest path routes to v.

Therefore, establishing the greedy path for the sink's responses is straightforward and memory-efficient while producing a short message route.

2.4 Energy-Constrained Flows

Next consider the situation where there is urgent data to be sent from one or more sources to a sink. For example, there could be an event requiring immediate attention, such as a natural disaster. Nodes generate data about it, but there is only limited time until the sink has to react. We model this as the following objective: Maximize the total information reaching the sink before a known point in time. Packets arriving later than this point cannot be considered in the decision process happening at the sink and are therefore useless. Furthermore, the nodes are equipped with non-rechargeable batteries. When a node runs out of energy due to excessive communication, it will shut down, thereby leaving the network.

To model the timing behavior of the sensor network accurately, we introduce an explicit notion of time: With each edge e, we associate a *transit time* $\tau_e \in \mathbb{N}$, defining how much time it takes a packet to travel e. This puts our problem into the context of Dynamic Network Flows [17]. We assume that a *time horizon* $T \in \mathbb{N}$ is given. Each edge $e \in E$ can be used in both directions. The flow on an edge is bounded by an *edge capacity* u_e, which applies to the sum of both directions. Let s be the source and t the sink. Let $\mathcal{P}^{st}$ be the set of all directed, simple s-t-paths in the network. The total transit time of a path $P \in \mathcal{P}^{st}$ is denoted by $\tau(P) := \sum_{e \in P} \tau_e$. We define $\tau_e(P)$ as the time a packet traveling P needs until it reaches the edge $e \in P$. We assume that sending one unit of flow from u to v over $uv \in E$ reduces C_u, the initial energy level of u, by c^{s}_{uv}, and C_v by c^{r}_{uv}. The aggregated cost $c^*_{v,P}$ denotes the total energy cost for a node $v \in V$ when one flow unit is sent over path P.

The *Energy-Constrained Dynamic Flow* problem (ECDF) can be formulated as follows, where $x_P(\theta)$ denotes the flow starting to travel P at time θ:

$$\max \sum_{P \in \mathcal{P}^{st}} \sum_{\theta=0}^{T-\tau(P)} x_P(\theta) \tag{1}$$

$$\text{s.t.} \sum_{\substack{P \ni e: \\ 0 \leqslant \theta - \tau_e(P) \leqslant T - \tau(P)}} x_P(\theta - \tau_e(P)) \leqslant u_e \quad \forall e \in E, \theta = 0, \ldots, T \tag{2}$$

$$\sum_{P \ni v} \sum_{\theta=0}^{T-\tau(P)} c^*_{v,P} x_P(\theta) \leqslant C_v \qquad \forall v \in V \tag{3}$$

$$x_P(\theta) \geqslant 0 \qquad \forall P \in \mathcal{P}^{st}, \theta = 0, \ldots, T - \tau(P)\,. \tag{4}$$

It should be noted that our treatment [18] of the problem is the only one addressing energy-constrained dynamic flows so far, so there are many problem variants left unsolved.

An important special case of the problem is 1-ECDF, where we assume all transit times to be 1. This case reflects the store-and-forward message passing in

WSNs accurately. We showed that two important problem variants are hard [18]:

Theorem 4. *Finding an integral solution for 1-ECDF is* NP*-hard. The original ECDF with arbitrary transit times* $\tau \in \mathbb{N}^E$ *is also* NP*-hard.*

We were not able to decide whether the fractional 1-ECDF problem is hard. We presented some evidence that there may not be a polynomial-size encoding scheme for optimal solutions [18]. If this could be verified, it would prove that the problem is indeed PSPACE-hard. Unfortunately, this remains an open problem.

Our most important result on this problem is the following:

Theorem 5. *1-ECDF admits a distributed FPTAS.*

Proof sketch. Consider a solution to 1-ECDF, where for given P, $x_P(\theta)$ is constant over all feasible θ. Such a solution is called *temporally repeated.* It can be shown that temporally repeated flows are $(1-\varepsilon)$-approximations to optimal flows, when the time horizon T is larger than $\frac{1}{\varepsilon}|V|$. We consider LP formulations for temporally repeated flows and large time horizons as well as for 1-ECDF with small time horizons. Both use an exponential number of variables, yet only a polynomial number of constraints. Furthermore, both formulations describe fractional packing problems. Such problems can be solved using the algorithmic framework by Garg and Könemann [19], which provides a $(1-\varepsilon)^2$-approximation. This algorithm does a number of flow augmentation steps that are controlled by solving the dual separation problem for the LP formulation, where the important twist is to use exponentially growing dual weights. It can be shown that each such step corresponds to finding a shortest path $P \in \mathcal{P}^{st}$ w.r.t. a nonlinear path cost function. Finding such a shortest path can be $(1-\varepsilon)$-approximated by using simplified linear path costs. Our algorithm therefore provides a $(1-\varepsilon)^4$-approximation in polynomial time, by iteratively augmenting flow over shortest paths. The algorithm is easily distributed and requires little more storage at the nodes than the solution itself. □

3 Shawn: A Customizable Sensor Network Simulator

Software for WSNs must be thoroughly tested prior to real-world deployments, because sensor nodes do not offer convenient debugging interfaces and are typically inaccessible after deployment. Furthermore, successfully designing algorithms and protocols for WSNs requires a deep understanding of these complex distributed networks. To achieve these goals, three different approaches are commonly used: analytic methods, real-world experiments, and computer simulations.

Analytic methods are typically not well-suited to support the development of complete WSN applications. Despite their expressiveness and generality, it is difficult to grasp all details of such complex, distributed applications in a purely formal manner.

Real-world experiments are an attractive option as they are a convincing means to demonstrate that an application is able to accomplish a specific task in practice—if the technology is already available. However, due to the unpredictable environmental influences it is hard to reproduce results or to isolate sources of errors.

Computer simulations are a promising means to tackle the task of algorithm and protocol engineering for WSNs. Existing simulation tools such as Ns-2 [20], OMNeT++ [21], TOSSIM [22], or SENSE [23], reproduce real-world effects inside a simulation environment and therefore mitigate required efforts for real-world deployments. However, the high level of detail provided by these tools obfuscates and misses another, much more crucial issue: The large number of factors that influence the behavior of the whole network renders it nearly impossible to isolate a specific parameter of interest.

For example, consider the development of a routing protocol. The cause for low throughput is not clear at first sight, as the sources for the error are manifold: the MAC layer might be faulty; cross-traffic could limit the bandwidth; or the routing protocol's algorithm is not yet optimal. Therefore, it is not only sufficient to simulate a high number of nodes with a high level of detail. Instead, developers require the ability to focus on the actual research problem.

When designing algorithms and protocols for WSN it is important to understand the underlying structure of the network—a task that is often one level above the technical details of individual nodes and low-level effects. There is certainly some influence of communication characteristics. From the algorithm's point of view, there is no difference between a simulation of low-level networking protocols and the alternative approach of using well-chosen random distributions on message delay and loss. Thus, using a detailed simulation may lead to the situation where the simulator spends much processing time on producing results that are of no interest at all.

To improve this situation, we propose a novel simulation tool: Shawn [24, 25, 26]. The central idea of Shawn is to replace low-level effects with abstract and exchangeable models so that simulations can be used for huge networks in reasonable time, while keeping the focus on the actual research problem. In the following, Section 3.1 discusses fundamental design goals of Shawn while Section 3.2 shows how these goals reflect themselves in Shawn's architecture. Finally, Section 3.3 compares Shawn's performance with two other prominent simulation tools (Ns-2 and TOSSIM).

3.1 Design Goals

Shawn differs in various ways from the above-mentioned simulation tools, while the most notable difference is its focus. It does not compete with these simulators in the area of network stack simulation. Instead, Shawn emerged from an algorithmic background. Its primary design goals are:

- Simulate the effect caused by a phenomenon, not the phenomenon itself.
- Scalability and support for extremely large networks.
- Free choice of the implementation model.

Simulate the effects. As discussed above, existing simulation tools perform a complete simulation of the MAC layer, including radio propagation properties such as attenuation, collision, fading and multi-path propagation. A central design guideline of Shawn is to *simulate the effect caused by a phenomenon, and not the phenomenon itself.* Shawn therefore only models the effects of a MAC layer for the application (e.g., packet loss, corruption and delay).

This has several implications on the simulations performed with Shawn. On the one hand, they are more predictable and there is a performance gain, because such a model can be implemented very efficiently. On the other hand, this means that Shawn is unable to provide the same detail level that, e.g., Ns-2 provides with regard to physical-layer or packet-level phenomena. However, if the model is chosen well, the effects for the application are virtually the same.

It must be mentioned though that the interpretation of obtained results must take the properties of the individual models into account. If, for instance, a simplified communication model is used to benchmark the results of a localization algorithm, the quality of the obtained solution remains unaffected. However, the actual running time of the algorithm is not representative for real-world environments, because no delay or loss occurs.

Scalability. One central goal of Shawn is to support numbers of nodes that are orders of magnitudes larger than the currently existing simulators. By simplifying the structure of several low-level parameters, their time-consuming computation can be replaced by fast substitutes, as long as the interest in the large-scale behavior of the system focuses on unaffected properties. A direct benefit of this paradigm is that Shawn can *simulate vast networks.*

To enable a fast simulation of these scenarios, Shawn can be custom-tailored to the problem at hand by selecting appropriate configuration options. This enables developers to optimize the performance of Shawn specifically for each single scenario. For example, a scenario without any mobility can be treated differently than a scenario where sensor nodes are moving.

Free model choice. Shawn supports a multi-stage development cycle, where developers can *freely choose the implementation model.* Using Shawn, they are not limited to the implementation of distributed protocols. The rationale behind this approach is that—given a first idea for a novel algorithm—the next natural step is not the design of a fully distributed protocol. In fact, it is more likely to perform a structural analysis of the problem at hand. To get a better understanding of the problem in this phase, it may be helpful to look at some exemplary networks to analyze their structure and the underlying graph representation. This allows developers to start from an initial idea and gradually leads to a fully distributed protocol.

3.2 Architecture

Shawn's architecture comprises three major parts (see Figure 6): *Models*, *Sequencer* and *Simulation Environment.*

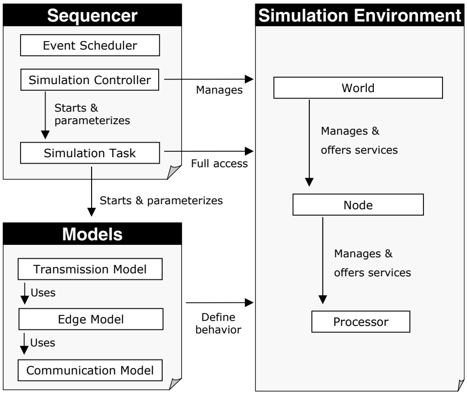


Fig. 6. High-level architecture of Shawn and overview of its core components

Every aspect of Shawn is influenced by one or more *Models*, which are the key to its flexibility and scalability. The *Sequencer* is the central coordinating unit in Shawn as it configures the simulation, executes tasks sequentially and controls the simulation. The *Simulation Environment* is the home for the virtual world in which the simulated sensor nodes reside. In the following, the functionality of these core components is described in detail.

Models. Shawn distinguishes between *models* and their respective implementations. A model is the interface used by Shawn to control the simulation without any knowledge on what a specific implementation may look like. Shawn maintains a repository of model implementations that can be used to compose simulation setups by selecting the desired behaviors. The implementation of a model may be simplified and fast, or it could provide close approximations to reality. This enables the user to select the most appropriate implementation of each model to fine-tune Shawn's behavior for a particular simulation task.

As depicted in Figure 6, three models form the foundation of Shawn: *Communication Model*, *Edge Model* and *Transmission Model*. In the following, these models and their already included implementations are explained in detail. Other models of minor importance are briefly introduced at the end of this section.

Communication Model. Whenever a simulated sensor node in Shawn transmits a message, the potential receivers of this message must be identified by the simulator. Note that this does not determine the properties of individual transmissions but defines whether two nodes can communicate at all. This question is answered by implementations of the *Communication Model*.

By implementing an instance of such a model, arbitrary communication patterns can be realized. Shawn comes with a set of different *Communication Model* implementations, such as the Unit Disk Graph Model, in which two nodes can communicate bidirectionally if the Euclidean distance d between the nodes is less than r_{max}; or the *Radio Irregularity Model* (RIM, [27, 28]), which is based on real-world experiments.

The *Edge Model* provides a graph representation of the network. The simulated nodes are the vertices of the graph, and an edge between two nodes is added whenever the *Communication Model* returns `true`. To assemble this graph representation, the *Edge Model* repeatedly queries the *Communication Model.* It is therefore possible to access the direct neighbors of a node, the neighbors of the neighbors, and so on. This is used by Shawn to determine the potential recipients of a message by iterating over the neighbors of the sending node. Simple centralized algorithms that need information on the communication graph can be implemented very efficiently as in contrast to other simulation tools, no messages must be exchanged that serve as probes for neighboring nodes.

Depending on the application's requirements and its properties, different storage models for these graphs are needed. For instance, mobile scenarios require different storage models than static scenarios. In addition, simulations of relatively small networks may allow storing the complete neighborhood of each node in memory. Conversely, huge networks will impose impractical demands for memory and hence supplementary edge models trade memory for runtime, e.g., by recalculating the neighborhood on each request or by caching a certain amount of neighborhoods. Accordingly, Shawn provides different *Edge Model* implementations.

Transmission Model. Whenever a node transmits a message, the behavior of the transmission channel may be completely different than for any other message transmitted earlier. For instance, cross traffic from other nodes may block the wireless channel or interference may degrade the channel's quality. To model these transient characteristics inside Shawn, the *Transmission Model* determines the properties of an individual message transmission. It can arbitrarily delay, drop or alter messages. This means that a message may not reach its destination even if the *Communication Model* states that two nodes can communicate and the *Edge Model* lists these two nodes as neighbors.

Again, the choice of an implementation strongly depends on the simulation goal. In case that the runtime of an algorithm is not of interest but only its quality, a simple transmission model without delay, loss or message corruption is sufficient. Models that are more sophisticated could take contention, transmission time and errors into account at the cost of performance.

Shawn's built-in transmission models cover both abstract and close-to-reality implementations. The *Reliable* transmission model delivers all messages immediately, without loss or corruption to all neighboring nodes. *Random drop* discards messages with a given probability but it neither delays nor alters messages. Additionally, an implementation is available that models the effect of the well-known CSMA/CA [29,30] medium access scheme.

Sequencer. The sequencer is the control center of the simulation: It prepares the world in which the simulated nodes live, instantiates and parameterizes the implementations of the models as designated by the configuration input and controls the simulation. It consists of *Simulation Tasks*, the *Simulation Controller* and the *Event Scheduler.*

Simulation Tasks are pieces of code that are invoked from the configuration of the simulation supplied by the user. They are not directly related to the simulated application, but they have access to the whole simulation environment and are thus able to perform a wide range of tasks. Example uses are managing simulations, gathering data from individual nodes or running centralized algorithms.

Shawn exclusively uses tasks to expose its internal features to the user. A variety of tasks is included in Shawn that supports the creation and parameterization of new simulation worlds, nodes, routing protocols, random variables, etc. Even the actual simulation is triggered using a task.

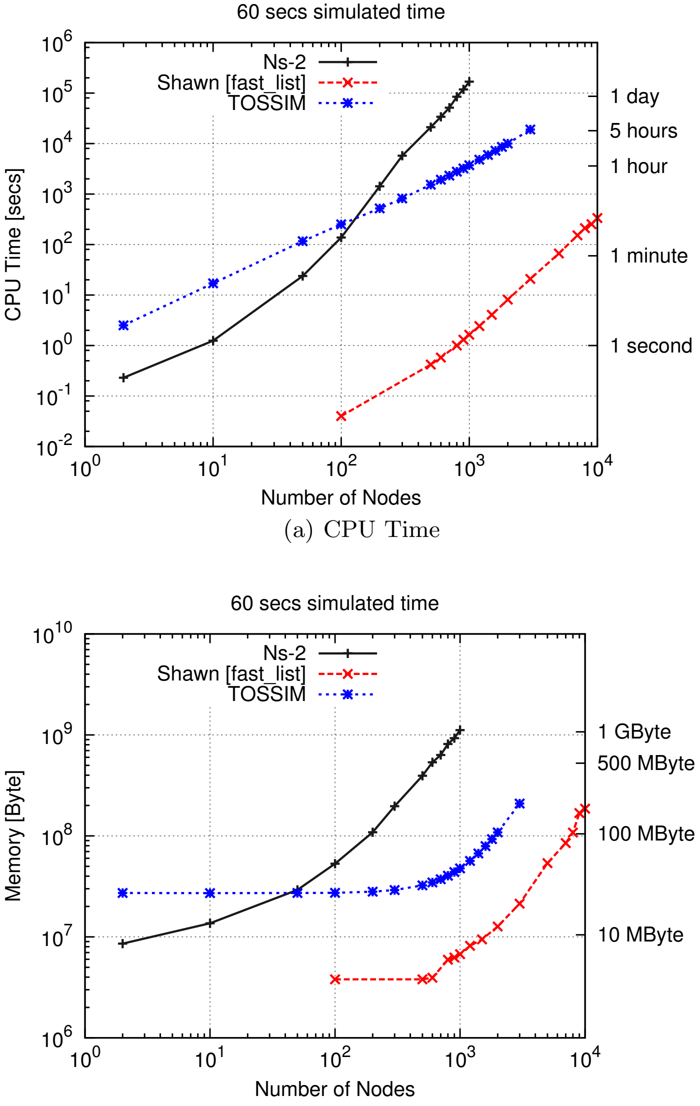
The *Simulation Controller* acts as the central repository for all available model implementations and runs the simulation by transforming the configuration input into parameterized invocations of *Simulation Tasks*. In doing so, it mediates between Shawn's simulation kernel and the user. In line with most other components of Shawn, the *Simulation Controller* can be customized by a developer to realize an arbitrary control over the simulation. The default implementation reads the configuration commands from a text file or the standard input stream, while a second one allows the use of Java scripts to steer the simulation. While providing the same functionality, it allows more complex constructs and evaluations already in the configuration file.

Event Scheduler. Shawn uses a discrete event scheduler to model time. The *Event Scheduler* is Shawn's time-keeping instance. Objects that need the notion of time can register with the Event Scheduler to be notified at an arbitrary point in time. The simulation always skips to the next event time and notifies the registered handlers. This process continues until all nodes signal either that they have powered down or until the maximum configured time has elapsed.

This has some performance advantages compared with traditional approaches that use fixed time intervals (such as a clock-tick every 1ms): First, handlers are notified only at the precise time that they have requested avoiding unnecessary calls to idle or waiting nodes that have no demand for processing.

Simulation Environment. The simulation environment is the home of the virtual world in which the simulation objects reside. As shown in Figure 6, the simulated *Nodes* reside in a single *World* instance. The Nodes themselves serve as a container for so-called *Processors*. Developers using Shawn implement their application logic as instances of these Processors. By decoupling the application inside a Processor from the Node, multiple applications can easily be combined in a single simulation run, without changing their implementations. For instance, one processor could implement an application-specific protocol while another processor gathers statistics data.

It is often the case that an algorithm requires input that is produced by multiple (potentially very complex) other algorithms. To avoid waiting for the same results of these previous steps repeatedly in every simulation run, Shawn offers the ability to attach type-safe information to Nodes, the World and the Simulation Environment. Simulation Tasks provide the ability to load tags from and to save tags to XML documents. A benefit of this concept is that it allows decoupling state variables from member variables in the user's simulation code. By this means, parts of a potentially complicated protocol can be replaced without code modification, because the internal state is stored in tags and not in member variables of a special implementation.



(a) CPU Time

(b) Memory

Fig. 7. Comparison of Shawn, Ns-2 and TOSSIM

3.3 Evaluation

This section evaluates the performance and adaptability of Shawn by comparing Shawn with Ns-2 and TOSSIM. Because the exchange of wireless messages is the key ingredient in wireless sensor networks, a simulator's ability to dispatch messages to their recipients determines the speed of simulations. In the following, measurements are presented that show the amount of memory and CPU time required to simulate a simple application that broadcasts a message every 250ms of simulated time.

The communication range of the sensor nodes is set to 50 length units and each simulation runs for 60 simulated time units. The size of the simulated area is 500x500 length units. A number of simulations with increasing node count were performed. Therefore, the network's density increases steadily as more nodes are added to the scenario. This application has been implemented for Ns-2, TOSSIM and Shawn. All simulation tools were used as supplied by the source repositories with maximal compiler optimization enabled. The simulations were run on standard, state-of-the-art i686 PCs.

Figure 7 depicts the required CPU time in seconds and the maximally used amount of RAM for the three simulation tools at different node counts. It should be noted that this kind of comparison is biased in favor of Shawn, because the two other simulators perform much more detailed computations to arrive at the same results; it should be seen primarily as an indication how application developers can benefit from using Shawn when these detailed results are not in the focus of interest. (Note that these and the following figures use a logarithmic scale on both axes.)

The first thing to notice is that Shawn outperforms both other simulation tools by orders of magnitude. Ns-2 hits the one-day barrier where Shawn is still finishing in less than one minute with a considerably smaller memory footprint. As mentioned above, this is because Ns-2 performs a very detailed simulation of lower layers such as the physical and the data-link layer, while Shawn simply dispatches the messages using a simplified model. Nearly the same situation applies to TOSSIM that simulates an underlying TinyOS-supported hardware platform. This clearly shows that Shawn excels in its specialty: the simulation of large-scale sensor networks with a focus on abstract, algorithmic considerations and high-level protocol development.

3.4 Conclusions and Future Work

The above measurements show that Shawn's central design goal (simulate the effect caused by a phenomenon, and not the phenomenon itself) indeed leads to a high scalability and performance when compared to traditional approaches to simulation. Therefore, developers must carefully select the simulation tool, depending on the application area. When detailed simulations of issues such as radio propagation properties or low-layer issues should be considered, Shawn is obviously not the perfect choice. This is where Ns-2 and TOSSIM offer the desired granularity. However, when developing algorithms and high-level protocols for WSNs, this level of detail often limits the expressiveness of simulations and blurs the view on the actual research problem. This is where Shawn provides the required abstractions and performance.

Shawn is currently in active development and used by several universities and companies to simulate wireless (sensor) networks. It was and continues to be an invaluable tool for over 20 research publications and more than 30 bachelor and master theses. Shawn is also the enabling means behind the development of algorithms and protocols for WSNs in the SWARMS [31], the SwarmNet [32] and the EU-funded FRONTS [33] and WISEBED [34] projects. It supports recent versions of Microsoft Windows and most Unix-like operating systems, such as Linux and Mac OS X. In addition, Shawn is easily portable to other systems with a decent C++-compiler. It is licensed under the liberal *BSD License*[1] and the full source code is available for download at `http://www.sourceforge.net/projects/shawn`.

References

1. Buschmann, C., Fekete, S.P., Fischer, S., Kröller, A., Pfisterer, D.: Koordinatenfreies Lokationsbewusstsein. IT- Information Technology 47, 70–78 (2005)
2. Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., Anderson, J.: Wireless sensor networks for habitat monitoring. In: ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002), Atlanta, GA (2002)
3. Szewczyk, R., Mainwaring, A., Polastre, J., Anderson, J., Culler, D.: An analysis of a large scale habitat monitoring application. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), pp. 214–226. ACM Press, New York (2004)
4. Zhang, P., Sadler, C.M., Lyon, S.A., Martonosi, M.: Hardware design experiences in ZebraNet. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), pp. 227–238. ACM Press, New York (2004)
5. Estrin, D., Govindan, R., Heidemann, J.: Embedding the Internet: Introduction. Commun. ACM 43(5), 38–41 (2000)
6. Kumar, V.: Sensor: the atomic computing particle. SIGMOD Rec. 32(4), 16–21 (2003)
7. Basu, A., Gao, J., Mitchell, J.S., Sabhnani, G.: Distributed localization using noisy distance and angle information. In: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC 2006), pp. 262–273. ACM Press, New York (2006)
8. Wagner, D., Wattenhofer, R. (eds.): Algorithms for Sensor and Ad Hoc Networks. LNCS, vol. 4621. Springer, Heidelberg (2007)
9. Fekete, S.P., Kröller, A., Pfisterer, D., Fischer, S., Buschmann, C.: Neighborhood-based topology recognition in sensor networks. In: Nikoletseas, S.E., Rolim, J.D.P. (eds.) ALGOSENSORS 2004. LNCS, vol. 3121, pp. 123–136. Springer, Heidelberg (2004)
10. Fekete, S.P., Kaufmann, M., Kröller, A., Zweig, K.A.: A new approach for boundary recognition in geometric sensor networks. In: Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG 2005), pp. 82–85 (2005)

[1] http://www.opensource.org/licenses/bsd-license.php

11. Kröller, A., Fekete, S.P., Pfisterer, D., Fischer, S.: Deterministic boundary recognition and topology extraction for large sensor networks. In: Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2006), pp. 1000–1009 (2006)
12. Choi, H.I., Choi, S.W., Moon, H.P.: Mathematical theory of medial axis transform. Pacific Journal of Mathematics 181, 57–88 (1997)
13. Sherbrooke, E.C., Patrikalakis, N.M., Wolter, F.E.: Differential and topological properties of medial axis transforms. Graphical Models and Image Processing 58(6), 574–592 (1996)
14. Kröller, A.: Algorithms for Topology-Aware Sensor Networks. Ph.D thesis, Braunschweig Institute of Technology (2008)
15. Förster, K.T.: Clusterbasierte Objektüberwachung in drahtlosen Sensornetzwerken: Lokal optimale Wege in der Ebene und ihre Anwendung in Sensornetzwerken. Diploma thesis, Braunschweig Institute of Technology (2007)
16. Fekete, S.P., Förster, K.T., Kröller, A.: Local routing in geometric cluster graphs (2009)
17. Ford, L.R., Fulkerson, D.R.: Constructing maximal dynamic flows from static flows. Operations Research 6, 419–433 (1958)
18. Fekete, S.P., Hall, A., Köhler, E., Kröller, A.: The maximum energy-constrained dynamic flow problem. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 114–126. Springer, Heidelberg (2008)
19. Garg, N., Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: Proc. FOCS, p. 300 (1998)
20. University of Southern California, Information Sciences Institute (ISI): Ns-2: Network simulator-2 (1995), http://www.isi.edu/nsnam/ns/
21. Varga, A.: OMNeT++: Objective modular network testbed in C++ (2007), http://www.omnetpp.org
22. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In: Proceedings of the First ACM Conference on Embedded Networked Sensor Systems, SenSys 2003 (2003), http://www.cs.berkeley.edu/~pal/research/tossim.html
23. Szymanski, B.K., Chen, G., Branch, J.W., Zhu, L.: SENSE: Sensor network simulator and emulator (2007), http://www.cs.rpi.edu/~cheng3/sense/
24. Kröller, A., Pfisterer, D., Buschmann, C., Fekete, S.P., Fischer, S.: Shawn: A new approach to simulating wireless sensor networks. In: Design, Analysis, and Simulation of Distributed Systems 2005 (DASD 2005), pp. 117–124 (2005)
25. Pfisterer, D., Fischer, S., Kröller, A., Fekete, S.: Shawn: Ein alternativer Ansatz zur Simulation von Sensornetzwerken. Technical report, 4. Fachgespräch Drahtlose Sensornetze der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme (2005)
26. Fekete, S.P., Kröller, A., Fischer, S., Pfisterer, D.: Shawn: The fast, highly customizable sensor network simulator. In: Proceedings of the Fourth International Conference on Networked Sensing Systems, INSS 2007 (2007)
27. Zhou, G., He, T., Krishnamurthy, S., Stankovic, J.A.: Impact of radio irregularity on wireless sensor networks. In: MobiSys 2004: Proceedings of the 2nd international conference on Mobile systems, applications, and services, pp. 125–138 (2004)
28. Zhou, G., He, T., Krishnamurthy, S., Stankovic, J.A.: Models and solutions for radio irregularity in wireless sensor networks. ACM Trans. Sen. Netw. 2(2), 221–262 (2006)
29. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed Systems: Concepts and Design, 4th edn. Addison Wesley, Reading (2005)

30. Tanenbaum, A.S., Steen, M.V.: Distributed Systems: Principles and Paradigms. Prentice Hall PTR, Englewood Cliffs (2001)
31. Fischer, S., Luttenberger, N., Buschmann, C., Koberstein, J.: SWARMS: Soft-Ware Architecture for Radio-based Mobile Self-organizing Systems, DFG SPP 1140 (2002), http://www.swarms.de
32. Fekete, S.P., Fischer, S., Kröller, A., Pfisterer, D.: SwarmNet: Algorithmen und Protokolle für Vernetzung und Betrieb großer Schwärme autonomer Kleinstprozessoren, DFG SPP 1126 (2003), http://www.swarmnet.de
33. Spirakis, P., et al.: Foundations of Adaptive Networked Societies of Tiny Artefacts (FRONTS), Research Academic Computer Technology Institute, 7th Framework Programme on Research, Technological Development and Demonstration (2007)
34. Fischer, S., et al.: Wireless sensornet test beds (WISEBED), Universität zu Lübeck, 7th Framework Programme on Research, Technological Development and Demonstration (2007)

Author Index