Lecture Notes in Computer Science 4368

*Commenced Publication in 1973*
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Thomas Erlebach  Christos Kaklamanis (Eds.)

# Approximation and Online Algorithms

4th International Workshop, WAOA 2006
Zurich, Switzerland, September 14-15, 2006
Revised Papers

Springer

Volume Editors

Thomas Erlebach
University of Leicester
Department of Computer Science
University Road, Leicester, LE1 7RH, UK
E-mail: t.erlebach@mcs.le.ac.uk

Christos Kaklamanis
University of Patras
Department of Computer Engineering and Informatics
26500, Rio, Patras, Greece
E-mail: kakl@ceid.upatras.gr

Library of Congress Control Number: 2006939787

CR Subject Classification (1998): F.2.2, G.2.1-2, G.1.2, G.1.6, I.3.5, E.1

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

ISSN 0302-9743
ISBN-10 3-540-69513-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-69513-4 Springer Berlin Heidelberg New York



Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11970125 06/3142 5 4 3 2 1 0

# Preface

The 4th Workshop on Approximation and Online Algorithms (WAOA 2006) focused on the design and analysis of algorithms for online and computationally hard problems. Both kinds of problems have a large number of applications from a variety of fields. WAOA 2006 took place at ETH Zurich in Zurich, Switzerland, during September 14–15, 2006. The workshop was part of the ALGO 2006 event that also hosted ESA, WABI, IWPEC, and ATMOS. The three previous WAOA workshops were held in Budapest (2003), Rome (2004), and Palma de Mallorca (2005). The proceedings of these previous WAOA workshops have appeared as LNCS volumes 2909, 3351 and 3879, respectively.

Topics of interest for WAOA 2006 were: algorithmic game theory, approximation classes, coloring and partitioning, competitive analysis, computational finance, cuts and connectivity, geometric problems, inapproximability results, mechanism design, network design, packing and covering, paradigms for design and analysis of approximation and online algorithms, randomization techniques, real-world applications, and scheduling problems. In response to the call for papers, we received 62 submissions. Each submission was reviewed by at least three referees, and the vast majority by at least four referees. The submissions were mainly judged on originality, technical quality, and relevance to the topics of the conference. Based on the reviews, the Program Committee selected 26 papers.

We are grateful to Andrei Voronkov for providing the EasyChair conference system, which was used to manage the electronic submissions, the review process, and the electronic PC meeting. It made our task much easier.

We would also like to thank all the authors who submitted papers to WAOA 2006 as well as the local organizers of ALGO 2006.

November 2006

Thomas Erlebach
Christos Kaklamanis

# Organization

## Program Co-chairs

| | |
|---|---|
| Thomas Erlebach | University of Leicester |
| Christos Kaklamanis | University of Patras |

## Program Committee

| | |
|---|---|
| Evripidis Bampis | University of Evry |
| Reuven Bar-Yehuda | Technion Haifa |
| Leah Epstein | University of Haifa |
| Thomas Erlebach | University of Leicester |
| Klaus Jansen | Universität Kiel |
| Christos Kaklamanis | University of Patras |
| Jochen Könemann | University of Waterloo |
| Danny Krizanc | Wesleyan University |
| Madhav Marathe | Virginia Tech |
| Seffi Naor | Microsoft Research and Technion, Israel |
| Alessandro Panconesi | University of Rome "La Sapienza" |
| Pino Persiano | Università di Salerno |
| Martin Skutella | Universität Dortmund |
| Roberto Solis-Oba | University of Western Ontario |
| Rob van Stee | Universität Karlsruhe |

## Additional Referees

| | | |
|---|---|---|
| Amjad Aboud | Sergio De Agostino | Laurent Gourvès |
| Ernst Althaus | Gianluca De Marco | Gregory Gutin |
| Eric Angel | Florian Diedrich | M.T. Hajiaghayi |
| Spyros Angelopoulos | György Dósa | Alex Hall |
| Vincenzo Auletta | Christoph Dürr | Han Hoogeveen |
| Nikhil Bansal | Pierre-Francois Dutot | Csanád Imreh |
| Gill Barequet | Alon Efrat | Yuval Ishai |
| Cristina Bazgan | Ran El-Yaniv | Liran Katzir |
| Eli Ben-Sasson | Roee Engelberg | Rohit Khandekar |
| Jit Bose | Guy Even | Samir Khuller |
| Niv Buchbinder | Lene M. Favrholdt | Stavros Kolliopoulos |
| Alberto Caprara | Dimitris Fotakis | Goran Konjevod |
| Deepti Chafekar | Martin Fürer | Alexander Kononov |
| JiangZhuo Chen | Stefan Funke | Guy Kortsarz |

Sven O. Krumke
V.S. Anil Kumar
Christian Laforest
Asaf Levin
Matthew Macauley
Ionnis Milis
Jérôme Monnot
Shlomo Moran
Pat Morin
Petra Mutzel
Lata Narayanan
Tom O'Connell
Ojas Parekh
Marco Pellegrini
Kirk Pruhs
Dror Rawitz
Joachim Reichel
Yossi Richter
Guido Schäfer
Heiko Schilling
Roy Schwartz
Ulrich M. Schwarz
Danny Segev
Hadas Shachnai
Sunil Shende
Gennady Shmonin
Mohit Singh
René Sitters
Alexander Souza
Mauro Sozio
S. S. Ravi
Nicolas Stier
Tami Tamir
Orestis A. Telelis
Nicolas Thibault
Shripad Thite
Ralf Thöle
Alessandro Tiberi
Eric Torng
Denis Trystram
Carmine Ventre
Tjark Vredeveld
Oren Weimann
Prudence Wong
Michal Ziv-Ukelson
Vadim Zverovich

# Table of Contents

# Approximation Algorithms for Scheduling Problems with Exact Delays*

Alexander A. Ageev and Alexander V. Kononov

Sobolev Institute of Mathematics, pr. Koptyuga 4, Novosibirsk, Russia
{ageev,alvenko}@math.nsc.ru

**Abstract.** We give first constant-factor approximations for various cases of the coupled-task single machine and two-machine flow shop scheduling problems with exact delays and makespan as the objective function. In particular, we design 3.5- and 3-approximation algorithms for the general cases of the single-machine and the two-machine problems, respectively. We also prove that the existence of a $(2-\varepsilon)$-approximation algorithm for the single-machine problem as well as the existence of a $(1.5-\varepsilon)$-approximation algorithm for the two-machine problem implies P=NP. The inapproximability results are valid for the cases when the operations of each job have equal processing times and for these cases the approximation ratios achieved by our algorithms are very close to best possible: we prove that the single machine problem is approximable within a factor of 2.5 and the two-machine problem is approximable within a factor of 2.

## 1 Introduction

In this paper we consider two scheduling problems with exact delays. In both problems a set $J = \{1, \ldots, n\}$ of independent jobs is given. Each job $j \in J$ consists of two operations with processing times $a_j$ and $b_j$ separated by a given intermediate delay $l_j$, which means that the second operation of job $j$ must start processing exactly $l_j$ time units after the first operation of job $j$ has been completed. In the single machine problem all operations are executed by a single machine. In the two-machine (flow shop) problem the first operations are executed by the first machine and the second ones by the second one. It is assumed that at any time no machine can process more than one operation and no preemptions are allowed in processing of any operation. The objective is to minimize the makespan (the schedule length). Extending the standard three-field notation scheme introduced by Graham et al. [6] we denote the single machine problem by $1 \mid \text{exact } l_j \mid C_{\max}$ and the two-machine problem by $F2 \mid \text{exact } l_j \mid C_{\max}$.

The problems with exact delays arise in command-and-control applications in which a centralized commander distributes a set of orders (associated with the first operations) and must wait to receive responses (corresponding to the second

* Research supported by the Russian Foundation for Basic Research, projects 05-01-00960, 06-01-00255.

T. Erlebach and C. Kaklamanis (Eds.): WAOA 2006, LNCS 4368, pp. 1–14, 2006.

operations) that do not conflict with any other (for more extensive discussion on the subject, see [4,8]). Research papers on problem $1 \mid \text{exact } l_j \mid C_{\max}$ are mainly motivated by applications in pulsed radar systems, where the machine is a multifunctional radar whose purpose is to simultaneously track various targets by emitting a pulse and receiving its reflection some time later [2,5,7,4,8]. Coupled-task scheduling problems with exact delays also arise in chemistry manufacturing where there often may be an exact technological delay between the completion time of some operation and the initial time of the next operation.

### 1.1 Related Work

Farina and Neri [2] present a greedy heuristic for a special case of problem $1 \mid \text{exact } l_j \mid C_{\max}$. Izquierdo-Fuente and Casar-Corredera [5] develop a Hopfield neural network for the problem. Elshafei et al. [4] present a Lagrange relaxation algorithm based on a discretization of the time horizon. Orman and Potts [7] establish that the problem is strongly NP-hard even in some special cases. In particular, they prove it for $1 \mid \text{exact } l_j, a_j = b_j = l_j \mid C_{\max}$. Yu [9], [10] proves that the two machine problem $F2 \mid \text{exact } l_j \mid C_{\max}$ is strongly NP-hard even in the case of unit processing times, which implies that the single machine problem is strongly NP-hard in the case of unit processing times as well (see [10]). Ageev and Baburin [1] present non-trivial constant-factor approximation algorithms for both the single and two machine problems under the assumption of unit processing times. More specifically, in [1] it is shown that problem $1 \mid \text{exact } l_j, a_j = b_j = 1 \mid C_{\max}$ is approximable within a factor of 7/4 and problem $F2 \mid \text{exact } l_j\ a_j = b_j = 1 \mid C_{\max}$, within a factor of 3/2.

### 1.2 Our Results

In this paper we present first constant-factor approximation algorithms for the general cases of $1 \mid \text{exact } l_j \mid C_{\max}$ and $F2 \mid \text{exact } l_j \mid C_{\max}$. We construct a 3.5-approximation algorithm for solving the single machine problem and 3-approximation algorithms for its special cases when $a_j \le b_j$, or $a_j \ge b_j$ for all $j \in J$. We also show that the latter algorithms provide a 2.5-approximation for the case when $a_j = b_j$ for all $j \in J$. Moreover, we prove that problem $1 \mid \text{exact } l_j \mid C_{\max}$ is not $(2-\varepsilon)$-approximable unless P=NP even in the case of $a_j = b_j$ for all $j \in J$. Addressing problem $F2 \mid \text{exact } l_j \mid C_{\max}$ we present a 3-approximation algorithm for the general case and show that it provides a 2-approximation for the cases when $a_j \le b_j$, or $a_j \ge b_j$ for all $j \in J$. Furthermore, we prove that the problem is not $(1.5-\varepsilon)$-approximable unless P=NP even in the case of $a_j = b_j$ for all $j \in J$. All designed algorithms can be implemented in $O(n \log n)$ time. The inapproximability results show that the approximation ratios achieved by our algorithms in the cases when $a_j = b_j$ for all $j \in J$ are very close to best possible: the single machine problem is approximable within a factor of 2.5 and not approximable within a factor of $(2-\varepsilon)$; the two machine problem is approximable within a factor of 2 and not approximable within a factor of $(1.5-\varepsilon)$. Approximability results established to date for the problems are summarized in Table 1.

**Table 1.** A summary of the approximability results

| problem | appr. factor | inappr. bound | ref. |
|---|---|---|---|
| $1 \mid \text{exact } l_j, \mid C_{\max}$ | 3.5 | $2-\varepsilon$ | this paper |
| $1 \mid \text{exact } l_j, a_j \le b_j \mid C_{\max}$ | 3 | $2-\varepsilon$ | this paper |
| $1 \mid \text{exact } l_j, a_j \ge b_j \mid C_{\max}$ | 3 | $2-\varepsilon$ | this paper |
| $1 \mid \text{exact } l_j, a_j = b_j \mid C_{\max}$ | 2.5 | $2-\varepsilon$ | this paper |
| $1 \mid \text{exact } l_j, a_j = b_j = 1 \mid C_{\max}$ | 1.75 | | [1] |
| $F2 \mid \text{exact } l_j \mid C_{\max}$ | 3 | $1.5-\varepsilon$ | this paper |
| $F2 \mid \text{exact } l_j, a_j \le b_j \mid C_{\max}$ | 2 | $1.5-\varepsilon$ | this paper |
| $F2 \mid \text{exact } l_j, a_j \ge b_j \mid C_{\max}$ | 2 | $1.5-\varepsilon$ | this paper |
| $F2 \mid \text{exact } l_j, a_j = b_j = 1 \mid C_{\max}$ | 1.5 | | [1] |

### 1.3 Basic Notation

For both problems an instance will be represented as a collection of triples $\{(a_j, l_j, b_j) : j \in J\}$ where $J = \{1, \ldots, n\}$ is the set of jobs, $a_j$ and $b_j$ are the lengths of the first and the second operations of job $j$, respectively and $l_j$ is the given delay between these operations. As usual, we assume that all input numbers are nonnegative integers. For a schedule $\sigma$ and any $j \in J$, denote by $\sigma(j)$ the starting time of the first operation of job $j$. As the starting times of the first operations uniquely determine the starting times of the second operations, any feasible schedule is uniquely specified by the collection of starting times of the first operations $\{\sigma(1), \ldots, \sigma(n)\}$. For a schedule $\sigma$ and any $j \in J$, denote by $C_j(\sigma)$ the completion time of job $j$ in $\sigma$; note that $C_j(\sigma) = \sigma(j) + l_j + a_j + b_j$ for all $j \in J$. The length of a schedule $\sigma$ is denoted by $C_{\max}(\sigma)$ and thus $C_{\max}(\sigma) = \max_{j \in J} C_j(\sigma)$. The length of a shortest schedule is denoted by $C^*_{\max}$.

The remainder of the paper is organized as follows. In Section 2 and 3 we describe and analyze the algorithms for the single machine and two-machine problems, respectively. Section 4 contains the inapproximability results.

## 2 Algorithms for the Single Machine Problem

In this section we describe and analyze approximation algorithms for the general and some special cases of the single machine problem.

### 2.1 Algorithms for Special Cases

We begin with presenting algorithm 1M$\le$ for the case when $a_j \le b_j$ for all $j \in J$.

Informally, the algorithm sorts the jobs in nonincreasing order of delays and then successively constructs segments of the output schedule, which we call blocks. An $s$-th block is the maximum possible subsequence of jobs $\{j_s, \ldots, j_{s+1} - 1\}$ that admits a feasible schedule in which the machine continuously processes the second operations of $j_1, \ldots, j_{s+1} - 1$. The performance analysis is based on the remarkable observation that the total idle time within each block except the first one can be evaluated via the processing times of the previously scheduled jobs.

Algorithm 1M$\leq$.

Phase I *(jobs ordering)*. Number the jobs in the following way:

$$a_1 + l_1 \geq a_2 + l_2 \geq \ldots \geq a_n + l_n \ . \tag{1}$$

Phase II *(constructing indices $j_s$)*. By examining the set of jobs in the order $j = 1, \ldots, n$ compute the indices $j_1 < j_2 < \ldots < j_r \leq n$ in the following way.

*Step* 1. Set $j_1 = 1$. If $\sum_{s=1}^{t-1} b_s \leq l_t$ for all $t = 2, \ldots, n$, then set $r = 1$, otherwise go to Step 2.

*Step* $k (k \geq 2)$. Set $j_k$ to be equal to the minimum index among indices $t > j_{k-1}$ such that $\sum_{s=j_{k-1}}^{t-1} b_s > l_t$. If $j_k = n$ or $\sum_{s=j_k}^{t-1} b_s \leq l_t$ for all $t = j_k + 1, \ldots, n$, then set $r = k$, otherwise go to Step $k + 1$.

Phase III *(constructing the schedule)*. Set $\sigma(j_1) = \sigma(1) = 0$. If $r > 1$, then for $s = 2, \ldots, r$ set

$$\sigma(j_s) = \sigma(j_{s-1}) + a_{j_{s-1}} + l_{j_{s-1}} + \sum_{k=j_{s-1}}^{j_s - 1} b_k \ . \tag{2}$$

For every $j \in J \setminus \{j_1, \ldots, j_r\}$, set

$$\sigma(j) = \sigma(j_s) + a_{j_s} + l_{j_s} - a_j - l_j + \sum_{k=j_s}^{j-1} b_k \tag{3}$$

where $s$ is the maximum index such that $j_s < j$.

**Example.** Consider the following instance of problem $1 \mid \text{exact } l_j, a_j \leq b_j \mid C_{\max}$ (the jobs are ordered according to Phase I):

$$\{(1, 6, 2), (2, 4, 3), (1, 5, 4), (1, 3, 2), (1, 3, 1), (1, 2, 3)\}.$$

Phase II finds that $i_1 = 1$, $i_2 = 4$, $i_3 = 6$, i. e., we have three blocks: $B_1 = \{1, 2, 3\}$, $B_2 = \{4, 5\}$, $B_3 = \{6\}$. Finally, Phase III computes the schedule $\sigma$: $\sigma(1) = 0$, $\sigma(2) = 3$, $\sigma(3) = 6$, $\sigma(4) = 16$, $\sigma(5) = 18$, $\sigma(6) = 23$ (see Fig. 1).

**Correctness and running time.** For convenience, set $j_{r+1} = n + 1$. Note that the set of jobs $J$ splits into $r$ disjoint subsets $B_s = \{j_s, \ldots, j_{s+1} - 1\}$, $s = 1, \ldots, r$ (we will further refer to them as *blocks*). The following lemma shows that algorithm 1M$\leq$ constructs a feasible schedule and describes its structure.

**Lemma 1.** *Let* $1 \leq s \leq r$.

- **(i)** *For any two jobs* $j', j'' \in B_s$ *such that* $j' < j''$, $\sigma(j'') \geq \sigma(j') + a_{j'}$.
- **(ii)** *For any job* $j \in B_s$, *the first operation of* $j$ *completes before starting the second operation of job* $j_s$.
- **(iii)** *The completion time of job* $j_{s+1} - 1$ *coincides with the starting time of the first operation of job* $j_{s+1}$.

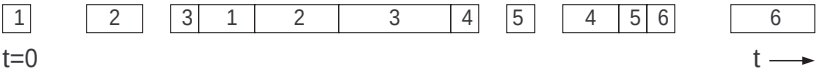



**Fig. 1.** The schedule constructed by algorithm 1M$\leq$

**(iv)** *Within the time interval*

$$[\sigma(j_s), \sigma(j_s) + a_{j_s} + l_{j_s} + \sum_{k=j_s}^{j_{s+1}-1} b_k]$$

*the machine executes both operations of each job in $B_s$ and only these operations.*

**(v)** *Within the time interval $[\sigma(j_s), \sigma(j_s) + a_{j_s} + l_{j_s}]$ the machine processes the first operations of all jobs in $B_s$ in the order $j_s, \ldots, j_{s+1} - 1$ and only these operations (with possible idle times).*

**(vi)** *Within the time interval*

$$[\sigma(j_s) + a_{j_s} + l_{j_s}, \sigma(j_s) + a_{j_s} + l_{j_s} + \sum_{k=j_s}^{j_{s+1}-1} b_k]$$

*the machine without idle times processes the second operations of all jobs in $B_s$ in the order $j_s, \ldots, j_{s+1} - 1$.*

*Proof.* First observe that (i), (iii), and (vi) imply (iv); (i), (ii), (iii) yield (v) as well.

Now let $j', j'' \in B_s$ and $j'' > j'$. By (3) we obtain that

$$\begin{aligned}\sigma(j'') - \sigma(j') &= \sigma(j_s) + a_{j_s} + l_{j_s} - a_{j''} - l_{j''} + \sum_{k=j_s}^{j''-1} b_k \\ &\quad - \Big(\sigma(j_s) + a_{j_s} + l_{j_s} - a_{j'} - l_{j'} + \sum_{k=j_s}^{j'-1} b_k\Big) \\ &= a_{j'} + l_{j'} - a_{j''} - l_{j''} + \sum_{k=j'}^{j''-1} b_k \ . \end{aligned} \tag{4}$$

By using $b_{j'} \geq a_{j'}$, (4), and (1) we obtain

$$\sigma(j'') - \sigma(j') - a_{j'} \geq b_{j'} + l_{j'} - a_{j''} - l_{j''} \geq a_{j'} + l_{j'} - a_{j''} - l_{j''} \geq 0 \ ,$$

which proves (i). Let $j \in B_s$. Then by the construction of $j_s$, $\sum_{k=j_s}^{j-1} b_k \leq l_j$. By (3) it follows that

$$\sigma(j) + a_j = \sigma(j_s) + a_{j_s} + l_{j_s} - l_j + \sum_{k=j_s}^{j-1} b_k \leq \sigma(j_s) + a_{j_s} + l_{j_s} \ ,$$

which yields (ii). Next we have

$$\begin{aligned} C_{j_{s+1}-1}(\sigma) &= \sigma(j_{s+1}-1) + a_{j_{s+1}-1} + l_{j_{s+1}-1} + b_{j_{s+1}-1} \\ \text{(by (3)} &= \sigma(j_s) + a_{j_s} + l_{j_s} - a_{j_{s+1}-1} - l_{j_{s+1}-1} \\ &\quad + \sum_{k=j_s}^{j_{s+1}-2} b_k + a_{j_{s+1}-1} + l_{j_{s+1}-1} + b_{j_{s+1}-1} \\ &= \sigma(j_s) + a_{j_s} + l_{j_s} + \sum_{k=j_s}^{j_{s+1}-1} b_k \\ \text{(by (2))} &= \sigma(j_{s+1}) \ , \end{aligned}$$

which establishes (iii). By (4) we have that

$$\sigma(j'') + a_{j''} + l_{j''} = \sigma(j') + a_{j'} + l_{j'} + \sum_{k=j'}^{j''-1} b_k \geq \sigma(j') + a_{j'} + l_{j'} + b_{j'} \ , \quad (5)$$

which means that the second operation of job $j''$ starts after the completion of job $j'$. Moreover, if $j'' = j' + 1$, then the inequality in (5) holds with equality, which means that the second operation of job $j''$ starts exactly after the completion of job $j'$ and thereby (vi) is verified. □

It is easy to see that the most time consuming part of the algorithm is the sorting on Phase I and so its running time is $O(n \log n)$.

**Approximation ratio.** First, note that $C^*_{\max}$ is at least the load of the machine and the maximum length of a job, i. e.,

$$C^*_{\max} \geq \max\{\sum_{j=1}^{n}(a_j + b_j), \max_{j \in J}(a_j + b_j + l_j)\} \ . \quad (6)$$

For $s = 1, \ldots, r$, set $H_s = a_{j_s} + l_{j_s} + \sum_{k=j_s}^{j_{s+1}-1} b_k$. By (iv) and (vi) of Lemma 1

$$C_{\max}(\sigma) = \sum_{s=1}^{r} H_s = \sum_{s=1}^{r}(a_{j_s} + l_{j_s}) + \sum_{j=1}^{n} b_j \ . \quad (7)$$

Recall that by the construction of $j_s$ for each $s \geq 2$, $\sum_{k=j_{s-1}}^{j_s-1} b_k > l_{j_s}$. Hence (7) implies that

$$\begin{aligned} C_{\max}(\sigma) &\leq \sum_{s=1}^{r} a_{j_s} + l_1 + \sum_{s=2}^{r} \sum_{k=j_{s-1}}^{j_s-1} b_k + \sum_{j=1}^{n} b_j \\ &\leq \Big(\sum_{s=1}^{r} a_{j_s} + \sum_{j=1}^{n} b_j\Big) + \sum_{j=1}^{n} b_j + l_1 \ . \end{aligned} \quad (8)$$

Thus by (6) for the case when $a_j \le b_j$ for all $j \in J$, we have $C_{\max}(\sigma) \le 3 \cdot C^*_{\max}$. In the case when $a_j = b_j$ for all $j \in J$, (6) implies that $\sum_{j=1}^{n} b_j \le \frac{1}{2} C^*_{\max}$, which together with (8) yields $C_{\max}(\sigma) \le \frac{5}{2} \cdot C^*_{\max}$. Summing up we obtain the following

**Theorem 1**

**(i)** *Algorithm 1M$\le$ finds a schedule of length at most thrice the length of a shortest schedule.*

**(ii)** *When applied to problem* $1 \mid$ *exact* $l_j, a_j = b_j \mid C_{\max}$ *algorithm 1M$\le$ finds a schedule of length at most* 2.5 *times the length of a shortest schedule.* □

Observe that problem $1 \mid \text{exact } l_j, a_j \ge b_j \mid C_{\max}$ reduces to problem $1 \mid \text{exact } l_j, a_j \le b_j \mid C_{\max}$ by the standard inverse of the time axis. Thus $1 \mid \text{exact } l_j, a_j \ge b_j \mid C_{\max}$ can be solved within a factor of 3 of the length of an optimal schedule as well.

### 2.2 Algorithm for the General Case

Let $I = \{(a_j, l_j, b_j) : j \in J\}$ be an instance of $1 \mid \text{exact } l_j \mid C_{\max}$.

ALGORITHM 1M.

1. If $\sum_{j=1}^{n} a_j > \sum_{j=1}^{n} b_j$, replace $I = \{(a_j, l_j, b_j) : j \in J\}$ by the symmetrical instance $\{(b_j, l_j, a_j) : j \in J\}$ (which is equivalent to the inverse of the time axis).
2. Form the new instance $I^* = \{(a_j, l_j, \overline{b}_j) : j \in J\}$ where $\overline{b}_j = \max\{a_j, b_j\}$ (note that $I^*$ is an instance of $1 \mid \text{exact } l_j, a_j \le b_j \mid C_{\max}$.)
3. By applying Algorithm 1M$\le$ to $I^*$ find a schedule $\sigma$.
4. If $\sum_{j=1}^{n} a_j \le \sum_{j=1}^{n} b_j$ output $\sigma$; otherwise output the inverse of $\sigma$.

**Running time.** It is clear that the running time of Algorithm 1M is of the same order as that of Algorithm 1M$\le$, i. e., the algorithm runs in time $O(n \log n)$.

**Approximation ratio.** Clearly, we may assume that

$$\sum_{j=1}^{n} a_j \le \sum_{j=1}^{n} b_j \ . \tag{9}$$

By (8) and the construction of $\overline{b}_j$, we have

$$\begin{aligned} C_{\max}(\sigma) &\le \Big( \sum_{s=1}^{r} a_{j_s} + \sum_{j=1}^{n} \overline{b}_j \Big) + \sum_{j=1}^{n} \overline{b}_j + l_1 \\ &= \Big( \sum_{s=1}^{r} a_{j_s} + \sum_{j=1}^{n} \max\{a_j, b_j\} \Big) + \sum_{j=1}^{n} \max\{a_j, b_j\} + l_1 \\ &\le \sum_{s=1}^{r} a_{j_s} + 2 \sum_{j=1}^{n} (a_j + b_j) + l_1 \ . \end{aligned}$$

Since by (9), $\sum_{s=1}^{r} a_{j_s} \leq \sum_{j=1}^{n} a_j \leq \frac{1}{2}\sum_{j=1}^{n}(a_j + b_j)$, it follows that

$$C_{\max}(\sigma) \leq \frac{5}{2}\sum_{j=1}^{n}(a_j + b_j) + l_1$$
$$(\text{by (6)}) \leq \frac{7}{2}C^*_{\max} \ .$$

Thus we arrive at the following

**Theorem 2.** *Algorithm 1M finds a schedule of length at most* 3.5 *times the length of a shortest schedule.* □

**Tightness.** We now present an example demonstrating that the approximation bound established for Algorithm $1M \leq$ cannot be improved on with respect to the lower bound (6). Let $x$ be a positive integer and $k = 2x - 1$. Consider the instance of problem $1 \mid \text{exact } l_j, a_j \leq b_j \mid C_{\max}$ consisting of one job $(1, k(x + 1), x)$ and $k$ identical jobs $(1, x - 1, x)$. So we have $n = k + 1$. It is easy to see that algorithm 1M$\leq$ outputs a schedule $\sigma$ consisting of $n$ blocks and thus

$$C_{\max}(\sigma) = 1 + k(x + 1) + x + k(1 + x - 1 + x) = 3kx + k + 1 + x = 3kx + 3x \ .$$

On the other hand, the lower bound (6) is

$$LB = \max\{1 + x + k + kx, 1 + x + k(x + 1)\} = (k + 1)(x + 1) \ .$$

Thus

$$\frac{C_{\max}(\sigma)}{LB} = \frac{3kx + 3x}{(k + 1)(x + 1)} = \frac{3(2x - 1)x + 3x}{2x(x + 1)} = \frac{6x^2}{2x^2 + 2x} \ ,$$

which tends to 3 as $x \to \infty$.

A similar construction shows that an approximation factor of 2.5 is tight with respect to the lower bound (6) for the case when $a_j = b_j$ for all $j \in J$.

## 3 Algorithm for the Two-Machine Problem

In the section we present a constant-factor approximation algorithm for the two-machine problem. We analyze its performance in general and some special cases of the problem. Theorem 5 shows that the approximation ratio of this rather simple algorithm is surprisingly close to best possible.

Algorithm 2M.

Phase I *(jobs ordering)*. Number the jobs in the following way:

$$a_1 + l_1 \leq a_2 + l_2 \leq \ldots \leq a_n + l_n \ . \tag{10}$$

Phase II *(constructing the schedule)*. Set $\sigma(1) = 0$. For $j = 2, \ldots, n$, set

$$\sigma(j) = \max\{\sigma(j - 1) + a_{j-1}, \sigma(j - 1) + b_{j-1} + a_{j-1} + l_{j-1} - a_j - l_j\} \ . \tag{11}$$

**Example.** Consider the following instance of problem $F2 \mid \text{exact } l_j \mid C_{\max}$ (the jobs are ordered according to Phase I):

$$\{(1,2,3),(3,1,1),(1,3,4),(2,3,2)\} \ .$$

Phase II computes the schedule $\sigma$ with $\sigma(1) = 0$, $\sigma(2) = 2$, $\sigma(3) = 5$, $\sigma(4) = 8$ (see Fig. 2).

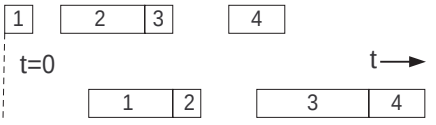



**Fig. 2.** The schedule constructed by algorithm 2M

**Correctness and running time.** By (11) for any $j = 2, \ldots, n$, $\sigma(j) \geq \sigma(j-1) + a_{j-1}$, which guarantees that the first operations of different jobs do not overlap. Moreover, by (11) for any $j = 2, \ldots, n$,

$$C_j(\sigma) - b_j = \sigma(j) + a_j + l_j \geq \sigma(j-1) + a_{j-1} + b_{j-1} + l_{j-1} = C_{j-1}(\sigma) \ , \quad (12)$$

which means that the second operation of job $j$ starts after the completion of the second operation of the previous job $j-1$, $j = 2, \ldots, n$. Therefore the second operations of different jobs do not overlap as well. Thus $\sigma$ is a feasible schedule.

It is clear that the algorithm can be implemented in $O(n \log n)$ time.

**Approximation ratio**

**Theorem 3**

**(i)** *Algorithm 2M finds a schedule of length at most thrice the length of a shortest schedule.*

**(ii)** *When applied to the special cases where $a_j \leq b_j$, or $a_j \geq b_j$ for all $j \in J$, algorithm 2M finds a schedule of length at most twice the length of a shortest schedule.*

*Proof.* Observe that $C^*_{\max}$ is at least the maximum load of machines and the maximum job length, i. e.,

$$C^*_{\max} \geq \max\{\max_{j \in J}(a_j + b_j + l_j), \sum_{j \in J} a_j, \sum_{j \in J} b_j\} \ . \quad (13)$$

By (12) job $n$ is the last job processed on machine 2, which means that the length of $\sigma$ coincides with the completion time of this job, i. e.,

$$C_{\max}(\sigma) = \sigma(n) + a_n + b_n + l_n \ . \quad (14)$$

Since $\sigma(1) = 0$, we have

$$\sigma(n) = \sum_{j=2}^{n} \big(\sigma(j) - \sigma(j-1)\big) \ . \tag{15}$$

By (10) and (11), for $j = 2, \ldots, n$,

$$\begin{aligned} \sigma(j) &\le \max\{\sigma(j-1) + a_{j-1}, \sigma(j-1) + b_{j-1}\} \\ &= \sigma(j-1) + \max\{a_{j-1}, b_{j-1}\} \ . \end{aligned}$$

By (15), it follows that $\sigma(n) \le \sum_{j=2}^{n} \max\{a_{j-1}, b_{j-1}\}$ and by (14),

$$C_{\max}(\sigma) \le \sum_{j=2}^{n} \max\{a_{j-1}, b_{j-1}\} + a_n + b_n + l_n \ . \tag{16}$$

By (13) it follows that in the case of arbitrary $a_j$ and $b_j$, $C_{\max}(\sigma) \le 3C^*_{\max}$ and in the case when $a_j \ge b_j$ or $b_j \ge a_j$ for all $j \in J$, $C_{\max}(\sigma) \le 2C^*_{\max}$. □

**Tightness.** As above we present an example validating that an approximation factor of 3 cannot be improved on with respect to the lower bound (13). Let $k$ be a positive integer. Consider the instance of $F2 \mid \text{exact } l_j \mid C_{\max}$ consisting of $k+1$ identical jobs $(1, k^2, k)$ and $k$ identical jobs $(k, k^2 - k + 1, 1)$, i. e., the total number of jobs $n = 2k + 1$. As $a_j + l_j = 1 + k^2$ for all $j \in J$, Phase I may index the jobs in the following alternating order:

$$(1, k^2, k), (k, k^2 - k + 1, 1), (1, k^2, k), (k, k^2 - k + 1, 1), \ldots, (1, k^2, k) \ .$$

It is easy to see that using this order Phase II computes the schedule $\sigma$ with $\sigma(2s + 1) = 2ks$ and $\sigma(2s) = k + 2ks$ for $s = 1, \ldots, k$ (the case of $k = 3$ is depicted in Fig. 3). Therefore the completion time of the last operation on the first machine is $2k^2 + 1$ and so

$$C_{\max}(\sigma) = 2k^2 + 1 + k^2 + k = 3k^2 + k + 1 \ .$$

On the other hand, the lower bound $LB = k^2 + 2k$. Thus

$$\frac{C_{\max}(\sigma)}{LB} = \frac{3k^2 + k + 1}{k^2 + 2k} \ ,$$

which tends to 3 as $k \to \infty$.

The tightness of an approximation factor of 2 for the case of $a_j = b_j$ for all $j \in J$ is an easy exercise.

## 4 Inapproximability Lower Bounds

In this section we establish inapproximability lower bounds for both problems. To this end we construct specific polynomial-time reductions from the following well-known NP-complete problem [3]:

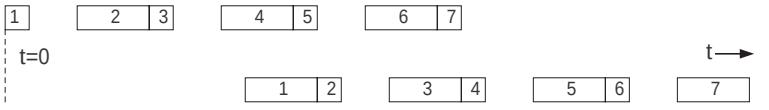



**Fig. 3.** The case of $k = 3$

PARTITION

**Instance:** Nonnegative numbers $w_1, \ldots, w_m$.

**Question:** Does there exist a subset $X \subseteq \{1, \ldots, m\}$ such that $\sum_{k \in X} w_k = S$ where $S = \frac{1}{2} \sum_{k=1}^{m} w_k$?

### 4.1 Problem 1 | exact $l_j$, $a_j = b_j$ | $C_{\max}$

Let $I$ be an instance of PARTITION. Define an instance $I^* = \{(a_j, l_j, b_j) : j \in J\}$ of problem $1 \mid \text{exact } l_j,\ a_j = b_j \mid C_{\max}$. Let $J = \{1, \ldots, m+3\}$ and

$$\begin{aligned} a_j = b_j = w_j, \quad l_j = (2q+3)S - w_j \quad &\text{for } j = 1, \ldots, m \ , \\ a_j = b_j = S, \quad l_j = (2q+4)S \quad &\text{for } j = m+1, m+2 \ , \\ a_{m+3} = b_{m+3} = qS, \quad l_{m+3} = 0 & \end{aligned}$$

where $q$ is a positive integer.

**Lemma 2**

**(i)** *If $\sum_{k \in X} w_k = S$ for some subset $X \subseteq \{1, \ldots, m\}$, then $C^*_{\max} = (2q+8)S$.*
**(ii)** *If $\sum_{k \in X} w_k \neq S$ for all $X \subseteq \{1, \ldots, m\}$, then $C^*_{\max} \geq (4q+3)S$.*

*Proof.*

(i). First, observe that since $(2q+8)S$ is the load of the machine, $C^*_{\max} \geq (2q+8)S$. Now we present a schedule $\sigma$ with $C_{\max}(\sigma) = (2q+8)S$. Set $\sigma(m+1) = 0$, $\sigma(m+2) = 2S$, $\sigma(m+3) = 4S$, and in an arbitrary order put the first operations of $j \in X$ within the interval $[S, 2S]$ and the first operations of $j \in \{1, \ldots, m\} \setminus X$ within the interval $[3S, 4S]$ (see Fig. 4). Then the second operations of jobs in $X$ and in $\{1, \ldots, m\} \setminus X$ will be processed in the same order within the vacant time intervals $[2qS + 4S, 2qS + 5S]$ and $[2qS + 6S, 2qS + 7S]$, respectively (see Fig. 4). Thus $C_{\max}(\sigma) = (2q+8)S$, as required.

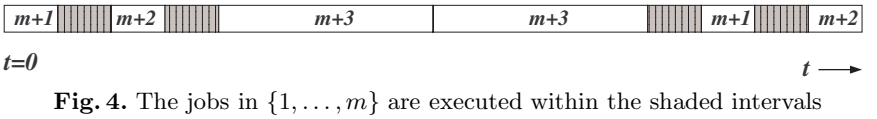



**Fig. 4.** The jobs in $\{1, \ldots, m\}$ are executed within the shaded intervals

(ii). Assume to the contrary that $C_{\max}(\sigma) < (4q+3)S$ for some feasible schedule $\sigma$ of $I^*$. We first claim that both operations of job $m+3$ are processed

between the first and second operations of each of the remaining jobs. Indeed, if job $m+3$ is not processed between the operations of some job in $\{m+1, m+2\}$, then

$$C_{\max}(\sigma) \geq 2qS + (2q+4)S = 4qS + 4S$$

and if job $m+3$ is not processed between the operations of some job in $\{1, \ldots, m\}$, then

$$C_{\max}(\sigma) \geq 2qS + (2q+3)S = 4qS + 3S \ .$$

Since the jobs $m+1$ and $m+2$ are identical, we may assume that $\sigma(m+1) < \sigma(m+2) < \sigma(m+3)$. Then by the claim and the construction of $I^*$,

$$\sigma(m+3) < C_{m+3}(\sigma) < C_{m+1}(\sigma) < C_{m+2}(\sigma) \ .$$

Since the first operations of all jobs in $\{1, \ldots, m+2\}$ are processed before time $\sigma(m+3)$,

$$\sigma(m+3) \geq 4S \ . \tag{17}$$

By the above claim, $\sigma(j) < \sigma(m+3) < C_{m+3}(\sigma) < C_j(\sigma)$ for all $j \in \{1, \ldots, m\}$. Assume that the first executable job is $j \in \{1, \ldots, m\}$. Hence $0 = \sigma(j) \leq \sigma(m+1)$. Then by the definition of job $j$,

$$C_{m+3}(\sigma) \leq C_j(\sigma) - w_j = \sigma(j) + 2w_j + 2qS + 3S - 2w_j = 2qS + 3S \ .$$

Since $C_{m+3}(\sigma) = \sigma(m+3) + 2qS$, it follows that $\sigma(m+3) \leq 3S$, contradicting the fact that the first operations of all jobs in $\{1, \ldots, m+2\}$ are processed before time $\sigma(m+3)$. Thus $m+1$ is the first executable job, i. e., $\sigma(m+1) = 0$. By a similar way it can be shown (this also follows from the time axis symmetry) that job $m+2$ is the last executable job. It follows that the second operations of all jobs in $\{1, \ldots, m\}$ are processed within the interval $[C_{m+3}(\sigma), C_{m+2}(\sigma) - S]$ and thus we have

$$C_{m+2}(\sigma) - C_{m+3}(\sigma) \geq 4S \ . \tag{18}$$

Let $T' = [\sigma(m+3) - 3S, \sigma(m+3)]$, $T'' = [C_{m+3}(\sigma), C_{m+3}(\sigma) + 3S]$. Then by the construction of $I^*$, for any $j \in \{1, \ldots, m\}$, $\sigma(j) \in T'$ and $C_j(\sigma) \in T''$. Next, by (17) we have that

$$C_{m+3}(\sigma) = \sigma(m+3) + 2qS \geq 2qS + 4S \ .$$

On the other hand, (18) implies that

$$\sigma(m+3) \leq \sigma(m+2) + 2S \ . \tag{19}$$

Thus the first operation of job $m+2$ is processed within interval $T'$ while (17) implies that the second operation of job $m+1$ is processed within interval $T''$. Therefore we may define the following subintervals of $T'$ and $T''$:

$$\begin{aligned} T_1 &= [\sigma(m+3) - 3S, \sigma(m+2)] \ , \\ T_2 &= [\sigma(m+2) + S, \sigma(m+3)] \ , \\ T_3 &= [C_{m+3}(\sigma), C_{m+1}(\sigma) - S] \ , \\ T_4 &= [C_{m+1}(\sigma), C_{m+3}(\sigma) + 3S] \ . \end{aligned}$$

As $|T_1| + |T_2| = |T_3| + |T_4| = 2S$ and the operations of all jobs in $\{1, \ldots, m\}$ are processed within $\bigcup_{k=1}^{4} T_k$, we have that $\sum_{j:\sigma_j \in T_k} w_j = |T_k|$ for $k = 1, 2, 3, 4$, where $|T_k|$ stands for the length of $T_k$. In particular, it follows that within interval $T_1$ the machine without idle times executes the first operations of some jobs in $\{1, \ldots, m\}$. Then the second operations of these jobs are executed within interval $[C_{m+3}(\sigma), C_{m+3}(\sigma) + |T_1|]$, which therefore cannot contain the second operation of job $m + 1$. This implies that

$$\begin{aligned} C_{m+3}(\sigma) + |T_1| &= C_{m+3}(\sigma) - \sigma(m+3) + 3S + \sigma(m+2) \\ &= 2qS + 3S + \sigma(m+2) \\ &\leq C_{m+1}(\sigma) - S \\ &= 2qS + 6S - S = 2qS + 5S \end{aligned}$$

or, equivalently, $\sigma(m + 2) \leq 2S$. By (19) it follows that $\sigma(m + 3) \leq 4S$, which together with (17) yields $\sigma(m + 3) = 4S$. Then

$$|T_3| = C_{m+1}(\sigma) - S - C_{m+3}(\sigma) = 2qS + 6S - S - 4S - 2qS = S$$

and thus $\sum_{j:C_j(\sigma) \in T_3} w_j = |T_3| = S$, which contradicts the assumption of (ii). □

The following is a straightforward corollary of the lemma.

**Theorem 4.** *The existence of a $(2 - \varepsilon)$-approximation algorithm for problem $1 \mid$ exact $l_j$, $a_j = b_j \mid C_{\max}$ implies P=NP.* □

### 4.2 Problem $F2 \mid$ exact $l_j$, $a_j = b_j \mid C_{\max}$

Let $I$ be an instance of PARTITION. Define an instance $I^* = \{(a_j, l_j, b_j) : j \in J\}$ of $F2 \mid$ exact $l_j$, $a_j = b_j \mid C_{\max}$ in the same way as in the above subsection, i. e., set $J = \{1, \ldots, m + 3\}$ and

$$\begin{aligned} a_j = b_j = w_j, \quad l_j = (2q+3)S - w_j \quad &\text{for } j = 1, \ldots, m \ , \\ a_j = b_j = S, \quad l_j = (2q+4)S \quad &\text{for } j = m+1, m+2 \ , \\ a_{m+3} = b_{m+3} = qS, \quad l_{m+3} = 0 & \end{aligned}$$

where $q$ is a positive integer. The proof of the next lemma is quite similar to that of Lemma 2.

**Lemma 3**

**(i)** *If $\sum_{k \in X} w_k = S$ for some subset $X \subset \{1, \ldots, m\}$, then $C^*_{\max} \leq (2q + 8)S$* (a schedule of length $(2q + 8)S$ is depicted in Fig. 5).

**(ii)** *If $\sum_{k \in X} w_k \neq S$ for all $X \subset \{1, \ldots, m\}$, then $C^*_{\max} \geq (3q + 3)S$.* □

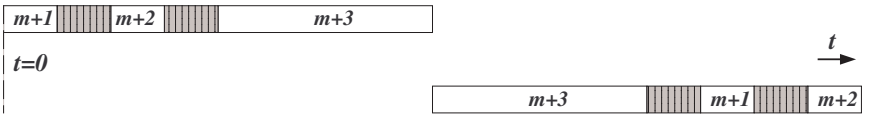



**Fig. 5.** The jobs in $\{1, \ldots, m\}$ are executed within the shaded intervals

As above we arrive at the following corollary:

**Theorem 5.** *The existence of a* $(1.5 - \varepsilon)$*-approximation algorithm for problem* $F2 \mid$ *exact* $l_j$, $a_j = b_j \mid C_{\max}$ *implies P=NP.* □

## References

1. A. A. Ageev and A. E. Baburin, Approximation Algorithms for the Single and Two-Machine Scheduling Problems with Exact Delays, to appear in Operations Research Letters.
2. A. Farina and P. Neri, Multitarget interleaved tracking for phased array radar, IEEE Proc. Part F: Comm. Radar Signal Process. 127 (1980) (4), 312–318.
3. M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, Freeman, San Francisco, CA, 1979.
4. M. Elshafei, H. D. Sherali, and J.C. Smith, Radar pulse interleaving for multi-target tracking, Naval Res. Logist. 51 (2004), 79–94.
5. A. Izquierdo-Fuente and J. R. Casar-Corredera, Optimal radar pulse scheduling using neural networks, in: IEEE International Conference on Neural Networks, vol. 7, 1994, 4588-4591.
6. R. L.Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 5 (1979), 287-326.
7. A. J. Orman and C. N. Potts, On the complexity of coupled-task scheduling, Discrete Appl. Math. 72 (1997), 141–154.
8. H. D. Sherali and J. C. Smith, Interleaving two-phased jobs on a single machine, Discrete Optimization 2 (2005), 348–361.
9. W. Yu, The two-machine shop problem with delays and the one-machine total tardiness problem, Ph.D. thesis, Technische Universiteit Eindhoven, 1996.
10. W. Yu, H. Hoogeveen, and J. K. Lenstra, Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. J. Sched. 7 (2004), no. 5, 333–348.

# Bidding to the Top: VCG and Equilibria of Position-Based Auctions

Gagan Aggarwal, Jon Feldman, and S. Muthukrishnan

Google, Inc.
76 Ninth Avenue, 4th Floor, New York, NY, 10011
1600 Amphitheatre Pkwy, Mountain View, CA, 94043
{gagana,jonfeld,muthu}@google.com

**Abstract.** Many popular search engines run an auction to determine the placement of advertisements next to search results. Current auctions at Google and Yahoo! let advertisers specify a single amount as their bid in the auction. This bid is interpreted as the maximum amount the advertiser is willing to pay per click on its ad. When search queries arrive, the bids are used to rank the ads linearly on the search result page. Advertisers seek to be high on the list, as this attracts more attention and more clicks. The advertisers pay for each user who clicks on their ad, and the amount charged depends on the bids of all the advertisers participating in the auction.

We study the problem of ranking ads and associated pricing mechanisms when the advertisers not only specify a bid, but additionally express their preference for positions in the list of ads. In particular, we study *prefix position auctions* where advertiser $i$ can specify that she is interested only in the top $\kappa_i$ positions.

We present a simple allocation and pricing mechanism that generalizes the desirable properties of current auctions that do not have position constraints. In addition, we show that our auction has an *envy-free* [1] or *symmetric* [2] Nash equilibrium with the same outcome in allocation and pricing as the well-known truthful Vickrey-Clarke-Groves (VCG) auction. Furthermore, we show that this equilibrium is the best such equilibrium for the advertisers in terms of the profit made by each advertiser. We also discuss other position-based auctions.

## 1 Introduction

In the sponsored search market on the web, advertisers bid on keywords that their target audience might be using in search queries. When a search query is made, an online (near-real time!) auction is conducted among those advertisers with matching keywords, and the outcome determines where the ads are placed and how much the advertisers pay. We will first review the existing auction model before describing the new model we study (a description can be found in Chapter 6 of [3]).

T. Erlebach and C. Kaklamanis (Eds.): WAOA 2006, LNCS 4368, pp. 15–28, 2006.

*Current Auctions.* Consider a specific query consisting of one or more *keywords*. When a user issues that search query, the search engine not only displays the results of the web search, but also a set of "sponsored links." In the case of Google, Yahoo, and MSN, these ads appear on a portion of the page near the right border, and are linearly ordered in a series of slots from top to bottom. (On Ask.com, they are ordered linearly on the top and bottom of the page).

Formally, for each search query, we have a set of $n$ advertisers interested in advertising. This set is usually derived by taking a union over the sets of advertisers interested in the individual keywords that form the query. Advertiser $i$ bids $b_i$, which is the maximum amount the advertiser is willing to pay for a click. There are $k < n$ positions available for advertisements. When a query for that keyword occurs, an online auction determines the set of advertisements, their placement in the positions, and the price per click each has to pay.

The most common auction mechanism in use today is the *generalized second-price* (GSP) auction (sometimes also referred to as the *next-price auction*). Here the ads are ranked in decreasing order of bid, and priced according to the bid of the next advertiser in the ranking. In other words, suppose wlog that $b_1 \geq b_2 \geq \ldots \geq b_n$; then the first $k$ ads are placed in the $k$ positions, and for all $i \in [1, k]$, bidder $i$ gets placed in position $i$ and pays $b_{i+1}$ per click.[1]

We note two properties ensured by this mechanism:

1. *(Ordering Property)* The ads that appear on the page are ranked in decreasing order of $b_i$.
2. *(Minimum Pay Property*) If a user clicks on the ad at position $i$, the advertiser pays the minimum amount she would have needed to *bid* in order to be assigned the position she occupies.

Search engine companies have made a highly successful business out of these auctions. In part, the properties above have dissuaded advertisers from trying to game the auction. In particular, the minimum-pay property ensures that an advertiser has no incentive to lower a winning bid by a small amount in order to pay a lower price for the same position. Still, the GSP auction is not truth-revealing, that is, an advertiser may be incentivized to bid differently than her true value under certain conditions [4].

Only recently have we obtained a detailed formal understanding of the properties of this auction. Authors in [1,2,4] have analyzed the auction in terms of its equilibria. They show that when the click-through rates are *separable*, i.e. the click-through rate of an ad at a given position is the product of an ad-specific factor and a position-specific factor, the GSP has a Nash equilibrium whose outcome is equivalent to the famous Vickrey-Clarke-Groves (VCG) mechanism [5,6,7] which is known to be truthful. [1,2] go on to show that this equilibrium is *envy-free*, that is, each advertiser prefers the current outcome (as it applies to her) to being placed in another position and paying the price-per-click being paid by the current occupant of the position. Further, among

[1] The Google auction actually ranks according to $w_i b_i$, for some weight $w_i$ related to the quality of the ad, and then sets the price for bidder $i$ to $w_{i+1} b_{i+1}/w_i$. All our results generalize to this "weighted" bid case as well.

all the envy-free equilibria, the VCG equilibrium is bidder-optimal; that is, for each advertiser, her price-per-click is minimum under this equilibrium. We note that when the click-through rates are separable, the outcome produced by the VCG mechanism has the ordering property. Authors in [4] also show that even when the click-through rates are arbitrary, there is a pricing method with the ordering property that is truthful. (This pricing method reduces to the VCG pricing method when the click-through rates are separable.) Furthermore, they show that the GSP has a Nash equilibrium that has the same outcome as their mechanism. Together, these results provide some understanding of the current auctions. That in turn provides confidence in the rules of the auction, and helps support the vast market for search keyword advertising.

*Emerging Position-Based Auctions.* As this market matures, advertisers are becoming increasingly sophisticated. For example they are interested in the relative performance of their ads and keywords, and so the search engines provide tools to track statistics. As advertisers learn when and where their ads are most effective, they need more control over their campaigns than is provided by simply stating a keyword and a bid.

One of the most important parameters affecting the performance of an advertisement is its position on the page. Indeed, the reason the auction places the ads in descending order on the page is that the higher ads tend to get clicked on more often than the lower ones. In fact, having an ad place higher on the page not only increases the chances of a click, it also has value as a branding tool, regardless of whether the ad gets clicked. Indeed, a recent empirical study by the Interactive Advertising Bureau and Nielsen//NetRatings concluded that higher ad positions in paid search have a significant brand awareness effect [8]. Because of this, advertisers would like direct control over the position of their ad, beyond just increasing the bid. Ideally, the search engine would conduct a more general auction that would take such position preferences into account; we refer to this as a *position-based auction.*

*Our Results.* In this paper, we initiate the study of position-based auctions where advertisers can impose position constraints. In particular, we study the most relevant case of *prefix* position constraints, inspired by the branding advertiser: advertiser $i$ specifies a position $\kappa_i$ and a bid $b_i$, which says that the advertiser would like to appear only in the top $\kappa_i$ positions (or not at all) and is willing to pay at most $b_i$ per click. Upon receiving bids from a set of $n$ such advertisers, the search engine must conduct an auction and place ads into $k$ positions while respecting the prefix constraints.

Our main results are as follows. We present a simple auction mechanism that has both the ordering and the minimum pay property, just like the current auctions. The mechanism is highly efficient to implement, taking near-linear time. Further, we provide a characterization of its equilibria. We prove that this auction has a Nash equilibrium whose outcome is equivalent in allocation and pricing to that of VCG. Additionally, we prove that this equilibrium is *envy-free* and that among all envy-free equilibria, this particular one is bidder-optimal.

Our results generalize those in [1,2], which proved the same thing for the GSP without position constraints. The main difficulty in generalizing these results lies in the fact that once you allow position constraints, the allocation function of VCG no longer obeys the ordering property, thus making it challenging to engineer an appropriate equilibrium. Our principal technical contributions are new structural properties of the VCG allocation that allow us to relate the VCG allocation with an auction that preserves the ordering property.

In the future, advertisers may want even more control over ad position. We discuss more general position-based auctions at the end of the paper.

## 2 Prefix Position Auction Mechanisms

Formally, the prefix position auction problem is as follows. There are $n$ advertisers for a search keyword. They submit bids $b_1, \ldots, b_n$ respectively. There are $k$ positions for advertisements numbered $1, \ldots, k$, top to bottom. Each advertiser $i \in \{1, \ldots, n\}$ also submits a cutoff position $\kappa_i \leq k$, and requires that their advertisements should not appear below position $\kappa_i$.

An auction mechanism consists of two functions:

- an *allocation* function that maps bids to a matching of advertisers to positions, as well as
- a *pricing* function that assigns a price per click $\text{ppc}_j$ to each position won by an advertiser. We restrict our attention to mechanisms where the prices always respect the bids; i.e., we have $\text{ppc}_j \leq b_i$ if $i$ is assigned to $j$.

A natural allocation strategy that retains the ordering property is as follows: rank the advertisers in decreasing order of $b_i$ as in GSP. Now, go through the ranking one position at a time, starting at the top; if you encounter an advertiser that appears below her bottom position $\kappa_i$, remove her from the ranking and move everyone below that position up one position, and continue checking down the rest of the list.

Two natural pricing strategies immediately come to mind here: **(1)** Set prices according to the subsequent advertiser in the ranking *before* any advertiser is removed, or **(2)** set prices according to the subsequent advertiser in the ranking *after* all the advertisers are removed (more precisely, the ones that appear below her position). It turns out that neither of these options achieves the minimum pay property as shown by the following examples. Assume for the sake of these examples that \$0.05 is the amount used to charge for the last position.

*Example 1.* Suppose we set prices before removing out-of-position advertisers. Now suppose we have the following ranges and bids where the number in parentheses is the position constraint $\kappa_i$:

```
A: (5) $5    B: (5) $4    C: (5) $3    D: (2) $2    E: (5) $1
```

We run the auction, and the order is (A, B, C, D, E). If we price now, the prices are (\$4, \$3, \$2, \$1, \$0.05). Bidder D gets removed and so we end up with (A, B, C, E),

and we charge ($4, $3, $2, $0.05). However, if bidder C had bid $1.50, which is below what she was charged, the auction would still have ended up as (A, B, C, E). Thus, the minimum pay property is violated by charging too much.

For a more intuitive reason why this is a bad mechanism, it would allow a form of "ad spam". Suppose a bidder sets her bottom cutoff to (2), but then bids an amount that would never win position one or two. In this case, she drives up the price for those that are later in the auction (e.g., competitors), at no risk and no cost to herself.

*Example 2.* Now suppose we set the prices after removing out-of-position advertisers, and we have the following bids and prefix constraints:

```
A: (5) $5    B: (5) $4    C: (2) $3    D: (5) $2
```

We run the auction, and the order is (A, B, C, D). Now we remove C and we get the order (A, B, D). We price according to this order and so the prices are ($4, $2, $0.05). Bidder B bid $4 and paid $2; however, if B changed her bid to $2.50, then bidder C would have gotten second position. Thus the minimum pay property is violated, but this time because we are charging too little.

As for intuition, this option opens up a possible "race for the bottom" situation. Suppose we have a group of bidders only interested in positions 1-4 (perhaps because those appear on the page without scrolling). The winners of the top three positions pay according to the competitive price for those top positions, but the winner of position 4 pays according to the winner of position 5, who could be bidding a much lower amount. Thus, these top bidders have an incentive to lower their prices so that they can take advantage of this bargain.

But now consider a third alternative, which will turn out to be the one that achieves the minimum-pay property: *For each advertiser that is allocated a particular position $j$, set the price according to the first advertiser that appears later in the original ranking that included $j$ in her range.* For an example of this pricing method, consider the situations from the examples above:

In Example 1, the advertisers would be ranked (A, B, C, D, E), and then (A, B, C, E) after removal. The price for A is set to $4, since B had position 1 in its range. Similarly, the price for B is set to $3 since C had position 2 in its range. The price for C is set to $1, however, since D did not include position 3 in its range. The price for C is set to $0.05.

In Example 2, the advertisers would be ranked (A, B, C, D) and after removal we get (A, B, D). The price for A is $4, but the price for B is now $3; even though C did not win any position, it was still a participant in the auction, and was bidding for position 2. The price for D is $0.05.

*Top-down Auction.* We now define an auction mechanism for prefix position constraints that is equivalent to the final proposal above, and is easily seen to have the minimum-pay property. Furthermore, this mechanism is exceedingly easy to implement, taking time $O(n \log n)$.

**Definition 1.** *The* top-down auction mechanism *works as follows: For each position in order from the top, iteratively run a simple second-price auction (with one winner) among those advertisers whose prefix range includes the position being considered. By a "simple second-price auction," we mean that the highest bidder in the auction is allocated the position, and pays a price-per-click equal to the second-highest bid. This winner is then removed from the pool of advertisers for subsequent auctions and the iteration proceeds.*

## 3 Analysis of the Top-Down Prefix Auction

We have found a natural generalization of GSP to use with prefix position constraints, and now we would like to know what properties this auction has. Since GSP is a special case, we already know that the auction is not truthful [4]. But from [2,1,4] we at least know something about the equilibria of GSP. It is natural to ask whether or not these results hold true in our more general setting.

In this section, we answer this in the affirmative, and prove that the top-down prefix auction has an "envy-free" Nash equilibrium whose outcome (in terms of allocation and pricing) is equivalent to that of VCG. ("Envy-freeness" is a stronger condition than is imposed by the Nash equilibrium, dictating that no bidder envies the allocation and price of any other bidder.) We go on to prove that this equilibrium is the bidder-optimal envy-free Nash equilibrium in the sense that it maximizes the "utility" (or profit) made by each advertiser.

*Definitions.* Each position $j$ has an associated *click-through rate* $c_j > 0$ which is the probability that a user will click on an ad in that position. Using the idea that higher positions receive more clicks, we may assume $c_1 > c_2 > \ldots > c_k$. To make the discussion easier, we will abuse this notation and say that an ad in position $j$ "receives $c_j$ clicks," even allowing $c_j > 1$ for some examples.

Each advertiser has a valuation $v_i$ it places on a click, as long as that click comes from one of its desired positions. Using the "branding" motivation, we assume a valuation of $-\infty$ if an ad even *appears* at a position below its bottom cutoff $\kappa_i$. Since $c_j > 0$ for all positions $j$, we can (equivalently) think of this as a valuation of $-\infty$ on a *click* below position $\kappa_j$. So, given some total price $p$ (for all the clicks) for a position $j$, the *utility* of bidder $i$ is defined as $u_i = c_j v_i - p$ if $j \leq \kappa_i$, and $-\infty$ otherwise.[2]

*The Vickrey-Clarke-Groves (VCG) Auction.* The VCG auction mechanism [5,6,7] is a very general technique that can be applied in a wide range of settings. Here we give its application to our problem. For a more general treatment, we refer the reader to Chapter 23 of [9].

[2] Note that we are making the assumption that click-through rates are dependent only on the position and not on the ad itself. Our results hold as long as the click-through rates are *separable*, i.e. the click-through rate of an ad at a given position is the product of a per-position factor and a per-advertiser factor. More general forms of click-through rate would require further investigation.

Let $\Theta$ represent the allocation of bidders to positions that maximizes the *total valuation* on the page; i.e., $\Theta$ is a matching $M$ of advertisers $i$ to positions $j$ that respects the position constraints ($j \leq \kappa_i$), and maximizes $\sum_{(i,j)\in M} v_i c_j$. Note that this assignment could also have empty slots, but they must be contiguous at the bottom end. The $\Theta$ allocation is the most "efficient" allocation, but an allocation function in an auction mechanism has access to the bids $b_i$ not the valuations $v_i$. So instead, the VCG allocation $M^*$ is the matching $M$ that maximizes $\sum_{(i,j)\in M} b_i c_j$.

Intuitively, the VCG price for a particular bidder is the total difference in others' valuation caused by that bidder's presence. To define this pricing function formally, we need another definition: Let $M^*_{-x}$ be the VCG allocation that would result if bidder $x$ did not exist. More formally, this allocation is the matching $M$ that does not include bidder $x$ and maximizes $\sum_{(i,j)\in M} b_i c_j$.

The VCG price for bidder $i$ in position $j$ is then $p_j = M^*_{-i} - M^* + c_j b_i$. (Here we are abusing notation and using $M^*$ and $M^*_{-i}$ to denote the total valuation of the allocation as well as the allocation itself.) Note that $p_j$ is a *total* price for all clicks at that position, not a per-click price. Only in the case that $b_i = v_i$ does the VCG mechanism actually successfully compute $\Theta$. However, it is well-known (see [9] for example) that the pricing method of VCG ensures that each bidder is incentivized to actually reveal their true valuation and set $b_i = v_i$. This holds *regardless of the actions of the other bidders,* a very strong property referred to as "dominant-strategy truthfulness." Thus in equilibrium, we get $b_i = v_i$, $M^* = \Theta$, and $M^*_{-i} = \Theta_{-i}$, where $\Theta_{-i}$ is the $\Theta$ allocation that would result if bidder $i$ did not exist.

For convenience, for the remainder of paper we rename the bidders by the slots to which they were assigned in $\Theta$, even when we are talking about the top-down prefix auction. The unassigned bidders are renamed to $(k+1, \ldots, n)$ arbitrarily. We will use $p_i = \Theta_{-i} - \Theta + c_i v_i$ to denote the VCG price (at equilibrium) for position (and bidder) $i$.

*Envy-Free Nash Equilibria and the GSP Auction.* The VCG mechanism is desirable because it has an equilibrium that results in the most efficient allocation according to the true valuations of the bidders. Furthermore this equilibrium occurs when each bidder actually reveals their true valuations. The GSP auction (without position constraints) does not have this second property, but in fact it does have the first: namely that it has an equilibrium whose allocation is the most efficient one (this was proved in [1,2,4]). Furthermore, this equilibrium also results in the same *prices* that result from VCG. This validates the GSP from an incentive-compatibility point of view, and shows that the ordering property does not preclude efficiency. This equilibrium also has the following property:

**Definition 2.** *An allocation and pricing is an* envy-free equilibrium *if each bidder prefers the current outcome (as it applies to her) to being placed in another position and paying the price-per-click being paid by the current occupant of the position.*

Moreover, among all envy-free Nash equilibria, this particular one is *bidder-optimal*, in the sense that it results in the lowest possible price for each particular advertiser. Note that in GSP, for a particular bidder, the only position for which envy-freeness is *not* implied by Nash is the position directly above.

### 3.1 Equilibrium in the Top-Down Auction

It is natural to ask if all these properties also hold true in the presence of position constraints. One of the difficulties in proving this comes from the fact that the VCG allocation no longer preserves the ordering property, as shown by the following simple example. Suppose advertiser `A` has bottom cutoff (2) and a bid of \$2, advertiser `B` has cutoff (1) and a bid of \$1, and we have $c_1 = 101$ and $c_2 = 100$. The VCG allocation gives position 1 to `B` and position 2 to `A`, for a total revenue of $\approx$ \$300. The top-down auction will give position 1 to `A` and position 2 will be unfilled. The revenue is equal to $\approx$ \$200.

Despite this, it turns out that there is an equilibrium of the top-down auction where bidders end up in the optimal allocation, which we prove in our main theorem:

**Theorem 1.** *In the top-down prefix auction, there exists a set of bids and stated position cutoffs such that*

*(a) each bidder is allocated to the same slot as she would be in the dominant-strategy equilibrium of VCG,*
*(b) the winner of each slot pays the same total price as she would have in the dominant-strategy equilibrium of VCG, and*
*(c) the bidders are in an envy-free Nash equilibrium.*

*Furthermore (d), for each advertiser, her utility under VCG outcomes is the maximum utility she can make under any envy-free equilibrium. In other words, a VCG outcome is a bidder-optimal envy-free equilibrium of the top-down auction.*

The remainder of this section is devoted to proving this theorem. The bids that satisfy this theorem are in fact quite simple: we set $b_i = p_{i-1}/c_{i-1}$ for all bidders $i$ assigned in $\Theta$. Thus, if we show that $b_1 > b_2 > \ldots > b_k$, we would get that the top-down auction assigns the bidders exactly like $\Theta$ and sets the same prices (modulo some technical details). This would prove (a) and (b) above.

*The chain.* To show that the bids are indeed decreasing, and to show (c), it turns out that we need to prove some technical lemmas about the difference between $\Theta$ and $\Theta_{-i}$ for some arbitrary bidder $i$. In $\Theta_{-i}$, some bidder $i'$ takes the place of $i$ (unless $i$ is in the last slot, in which case perhaps no bidder takes this slot). In turn, some bidder $i''$ takes the slot vacated by $i'$, etc., until either the vacated slot is the bottom slot $k$, or some previously unassigned bidder is introduced into the solution. We call this sequence of bidder movements ending at slot $i$ the "chain" of moves of $\Theta_{-i}$. Note that the chain has the property that it begins either with an unassigned bidder, or with the bidder from the last slot

and ends at slot $i$. If we consider the slots not on the chain, we claim that (wlog) the assignment does not change on these slots when we go from $\Theta$ to $\Theta_{-i}$. This is easily seen by substituting a purported better assignment on these slots back into $\Theta$. Note that this implies that $\Theta_{-i}$ has at most one new bidder (that wasn't in $\Theta$), and that no bidder besides $i$ that was assigned in $\Theta$ has dropped out. The chain is said to have *minimum length* if there is no shorter chain that achieves the same valuation as $\Theta_{-i}$. A *link* in this chain refers to the movement of a bidder $i$ from slot $i$ to some slot $i'$. We say that this is a *downward* link if $i' > i$; otherwise it is an upward link.

**Lemma 1.** *The minimum length chain for $\Theta_{-i}$ does not contain a downward link followed by an upward link.*

*Proof.* Suppose it does contain such a sequence. Then, some bidder $i_1$ moved from slot $i_1$ to slot $i_2 > i_1$, and bidder $i_2$ moved from slot $i_2$ to a slot $i_3 < i_2$. An alternate solution, and thus a candidate solution for $\Theta_{-i}$ is to have bidder $i_1$ move from slot $i_1$ to slot $i_3$, have bidder $i_2$ remain in slot $i_2$, and keep everything else the same. (Bidder $i_1$ can move to slot $i_3$ since $i_3 < i_2$ and $i_2$ is in range for bidder $i_1$ (by the fact that $i_1$ moved to $i_2$ in $\Theta_{-i}$).)

The difference between the two solutions is $c_{i_2}(v_{i_2} - v_{i_1}) + c_{i_3}(v_{i_1} - v_{i_2}) = (c_{i_3} - c_{i_2})(v_{i_1} - v_{i_2})$. We know $c_{i_3} > c_{i_2}$ since $i_3 < i_2$. We also know $v_{i_1} \geq v_{i_2}$ since otherwise $\Theta$ could switch bidders $i_1$ and $i_2$ (note again that bidder $i_1$ can move to slot $i_2$, since it did so in $\Theta_{-i}$). Thus the difference is non-negative, and so this alternate solution to $\Theta_{-i}$ has either greater valuation or a shorter chain. □

**Lemma 2.** *Let $x$ and $y$ be arbitrary bidders assigned to slots $x$ and $y$ in $\Theta$, where $x < y$. Then, (i) if slot $y$ is in the range of bidder $x$, we have $\Theta_{-y} \geq \Theta_{-x} + c_y(v_x - v_y)$, and (ii) $\Theta_{-x} \geq \Theta_{-y} + c_x(v_y - v_x)$.*

*Proof.* **(i)** Consider the assignment of bidder $y$ in $\Theta_{-x}$. Recall that for any $i$, all bidders besides $i$ present in $\Theta$ are also present in $\Theta_{-i}$. Thus $y$ is present somewhere in $\Theta_{-x}$. Note also that the minimum-length chain for $\Theta_{-x}$ ends at slot $x$, and so if $y$ is present in this chain, it cannot follow a downward link; otherwise the chain would contradict Lemma 1, since $x$ is above $y$. Thus we may conclude that $y$ ends up in position $y' \leq y$. Since slot $y$ is in range for bidder $x$ by assumption, we also have that $y'$ is in range for bidder $x$; thus we can construct a candidate solution for $\Theta_{-y}$ by replacing (in $\Theta_{-x}$) bidder $y$ with bidder $x$. We may conclude that $\Theta_{-y} \geq \Theta_{-x} + c_{y'}(v_x - v_y) \geq \Theta_{-x} + c_y(v_x - v_y)$.

**(ii)** This time we need to consider the assignment of $x$ in $\Theta_{-y}$. By the same logic as above, bidder $x$ is present somewhere, and if $x$ either stayed in the same place of moved up, we can replace $x$ with $y$ (in $\Theta_{-y}$) to get a candidate for $\Theta_{-x}$, and we are done. The only remaining case is when $x$ moves down in $\Theta_{-y}$ and this is a bit more involved.

Consider the section of the chain of $\Theta_{-y}$ from bidder $x$ to the end at bidder $y$ (who is below $x$). Since $x$ is on a downward link, and downward links cannot be followed by an upward link (Lemma 1), it must be the case that this section of the chain is entirely downward links. Let $x \to x_1 \to x_2 \to \ldots \to x_\ell \to y$ be this chain, and so we have $x < x_1 < x_2 \ldots < x_\ell < y$.

We write the assignment of $\Theta$ to these $\ell + 2$ places using the notation $[x, x_1, x_2, \ldots, x_\ell, y]$, and consider other assignments to these slots using the same notation. The solution $\Theta_{-y}$ assigns these slots as $[w, x, x_1, \ldots, x_\ell]$, where $w$ is the bidder before $x$ in the chain. For notational purposes define $x_{\ell+1} = y$.

Consider the following alternate solution constructed from $\Theta_{-y}$: change only the assignments to these special $\ell+2$ slots to $[w, x_1, \ldots, x_\ell, y]$. This is a candidate for $\Theta_{-x}$ and so by calculating the difference in valuation between this candidate solution and $\Theta_{-y}$ we get

$$\Theta_{-x} \geq \Theta_{-y} + \left( \sum_{i}^{\ell} v_{x_i} (c_{x_i} - c_{x_{i+1}}) \right) + c_y v_y - c_{x_1} v_x \tag{1}$$

Putting this aside for now, consider the following alternate solution for $\Theta$. Take the assignment in $\Theta$ and change the assignment to only those $\ell + 2$ positions to $[y, x, x_1, \ldots, x_\ell]$. This is feasible since $y$ moves up, and the remaining changes are identical to $\Theta_{-y}$. Since this solution must have valuation at most that of $\Theta$,

$$c_x v_y + c_{x_1} v_x + \sum_{1}^{\ell} v_{x_i} c_{x_{i+1}} \leq c_x v_x + \left( \sum_{1}^{\ell} v_{x_i} c_{x_i} \right) + c_y v_y$$

$$\iff c_x (v_y - v_x) \leq \left( \sum_{1}^{\ell} v_{x_i} (c_{x_i} - c_{x_{i+1}}) \right) + c_y v_y - c_{x_1} v_x$$

This, combined with (1), implies (ii). □

Now we are ready to prove the first part of our main theorem: that our bids give the same outcome as VCG, and are indeed an envy-free equilibrium.

**Proof of Theorem 1(a-c).** The bids of the equilibrium are defined as follows. For all bidders $i > 1$ assigned in $\Theta$, we set $b_i = p_{i-1}/c_{i-1}$. We set $b_1$ to any number greater than $b_2$. For all bidders assigned in $\Theta$, we set their stated cutoff to their true cutoff $\kappa_i$. If there are more than $k$ bidders, then for some bidder $\alpha$ that was not assigned in $\Theta$, we set $b_\alpha = p_k/c_k$, and set the stated cutoff of bidder $j$ to the bottom slot $k$. For all other bidders not in $\Theta$, we set their bid to zero, and their cutoff to their true cutoff.

Consider two arbitrary bidders $x$ and $y$ assigned in $\Theta$, where $x < y$. Using Lemma 2(ii), we get $\Theta_{-x} \geq \Theta_{-y} + c_x(v_y - v_x)$. Substituting for $\Theta_{-x}$ and $\Theta_{-y}$ using the definitions of $p_x$ and $p_y$, respectively, we get:

$$p_x - c_x v_x \geq p_y - c_y v_y + c_x (v_y - v_x) \iff \left( v_y - \frac{p_y}{c_y} \right) c_y \geq \left( v_y - \frac{p_x}{c_x} \right) c_x$$

Since $c_y < c_x$, we get $\frac{p_x}{c_x} > \frac{p_y}{c_y}$.

Since we chose $x$ and $y$ arbitrarily, we have just showed that $b_2 > \ldots > b_k$, and $b_k > b_\alpha$ if bidder $\alpha$ exists. We have $b_1 > b_2$ by definition, and all other bids are equal to zero. Thus the bids are decreasing in the VCG order, and so the

top-down auction will choose the same allocation as VCG. By construction, the top-down auction will also have the same prices as VCG.

It remains to show that this allocation and pricing is an envy-free equilibrium. Consider again two bidders $x$ and $y$ assigned in $\Theta$ with $x < y$. The utilities of $x$ and $y$ are $u_x = c_x v_x - p_x$ and $u_y = c_y v_y - p_y$. We must show that $x$ does not envy $y$, and that $y$ does not envy $x$.

If $y$ is out of range of bidder $x$, then certainly $x$ does not envy $y$. If $x$ is in range of bidder $y$, then by Lemma 2(i), we get $\Theta_{-y} \geq \Theta_{-x} + c_y(v_x - v_y)$. Substituting for $\Theta_{-y}$ and $\Theta_{-x}$ using the definitions of $p_y$ and $p_x$, we get

$$\begin{aligned} & \Theta + p_y - c_y v_y \geq \Theta + p_x - c_x v_x + c_y(v_x - v_y) \\ \iff & c_y v_x - p_y \leq c_x v_x - p_x = u_x. \end{aligned}$$

Thus $x$ does not envy $y$. Similarly, Lemma 2(ii) shows that $y$ does not envy $x$.

Now consider some bidder $z$ not assigned in $\Theta$. We must show that bidder $z$ does not envy any bidder that is assigned a slot in the desired range of $z$. Consider some such bidder $y$; replacing $y$ with $z$ creates a candidate for $\Theta_{-y}$. Thus we have $\Theta_{-y} \geq \Theta + c_y(v_z - v_y)$, which becomes $p_y = \Theta_{-y} - \Theta + c_y v_y \geq c_y v_z$. This implies that $z$ does not envy $y$. □

Now it remains to show the second part of Theorem 1, namely that among all envy-free equilibria, the one we define is optimal for each bidder. First we give a lemma showing that envy-freeness in the top-down auction implies that the allocation is the same as VCG. Then we use this to compare our equilibrium with an arbitrary envy-free equilibrium.

**Lemma 3.** *Any envy-free equilibrium of the top-down auction has an allocation with optimal valuation.*

*Proof.* For the purposes of this proof, we will extend any allocation of bidders to slots to place all $n$ bidders into "slots". For this, we will introduce dummy slots indexed by integers greater than $k$, with click-through rate $c_i = 0$. We index the bidders according to their (extended) allocation in $\Theta$.

For the purposes of deriving a contradiction, let $E, p$ be the allocation and pricing for an envy-free equilibrium of the top-down auction such that the valuation of $E$ is less than $\Theta$. Thus, $p_i$ refers to the price of slot $i$ in this envy-free equilibrium. Define a graph on $n$ nodes, one for each slot. For each bidder $i$, make an edge from $i$ to $j$, where $j$ is the slot in which bidder $i$ is placed in $E$; i.e., bidder $i$ is in slot $i$ in $\Theta$ and in slot $j$ in $E$. Note that this graph is a collection of cycles (a self-loop is possible, and is defined as a cycle).

Define the weight of an edge $(i, j)$ to be the change in valuation caused by bidder $i$ moving from slot $i$ in $\Theta$ to slot $j$ in $E$. So, we have that the weight of $(i, j)$ is equal to $v_i(c_j - c_i)$. Since the total change in valuation from $\Theta$ to $E$ is negative by definition, the sum of the weights of the edges is negative. This implies that there is a negative-weight cycle $Y$ in the graph, and so we have

$$\sum_{(i,j) \in Y} v_i(c_j - c_i) < 0. \tag{2}$$

By the fact that $E$ is envy-free, for each edge $(i, j)$, we also have that bidder $i$ would rather be in slot $j$ than in slot $i$ (under the prices $p$ imposed by the envy-free equilibrium). In other words, $v_i c_j - p_j \geq v_i c_i - p_i$. Rearranging and summing over the edges in $Y$, we get

$$\sum_{(i,j)\in Y} v_i(c_j - c_i) \geq \sum_{(i,j)\in Y} p_j - p_i = 0. \tag{3}$$

(The sum on the right-hand side equals zero from the fact that $Y$ is a cycle.) Equations (2) and (3) together give us a contradiction. □

Note that the profit of an advertiser $i$ is the same under all VCG outcomes, and is equal to the difference in valuation between $\Theta$ and $\Theta_{-i}$.

**Proof of Theorem 1(d).** Consider some envy-free equilibrium $E$ of the top-down auction. This equilibrium must have an allocation with optimal valuation (by Lemma 3). We will call this allocation $\Theta$. Let $\{p_i^E\}_i$ be the price of slot $i$ in this equilibrium; We will rename the bidders such that bidder $i$ is assigned to slot $i$ by allocation $\Theta$. Consider one such bidder $x$ assigned to slot $x$. Consider the chain $x_\ell \to x_{\ell-1} \to \ldots \to x_0 = x$ for $\Theta_{-x}$. (Here bidder $x_j$ moves from slot $x_j$ in $\Theta$ to slot $x_{j-1}$ in $\Theta_{-x}$.) By the fact that $E$ is envy-free, for all $j \in [0, \ell - 1]$ we have

$$\begin{aligned} & v_{x_{j+1}} c_{x_{j+1}} - p^E_{x_{j+1}} \geq v_{x_{j+1}} c_{x_j} - p^E_{x_j} \\ \iff & p^E_{x_j} \geq v_{x_{j+1}}(c_{x_j} - c_{x_{j+1}}) + p^E_{x_{j+1}}. \end{aligned}$$

(Each move is this chain is feasible, since it was made by $\Theta_{-x}$.) Composing these equations for $j = 0, \ldots, \ell - 1$, we get

$$p^E_x = p^E_{x_0} \geq v_{x_1}(c_{x_0} - c_{x_1}) + v_{x_2}(c_{x_1} - c_{x_2}) + \ldots + v_{x_\ell}(c_{x_{\ell-1}} - c_{x_\ell})$$

But note that each term of the right-hand side of this inequality represents the difference in valuation for a bidder on the chain of $\Theta_{-x}$. Thus the sum of these terms is exactly the VCG price $p_x$, and we have $p^E_x \geq p_x$. Hence, the profit of advertiser $x$ under equilibrium $E$ is no less than her profit under VCG. □

## 4 Concluding Remarks

The generalized second-price auction has worked extraordinarily well for search engine advertising. We believe that the essential properties of this auction that make it a success are that it preserves the ranking order inherent in the positions, and that it is stable in the sense that no bidder has an incentive to change her bid by a small amount for a small advantage. We have given a simple new prefix position auction mechanism that preserves these properties and has the same equilibrium properties as the regular GSP.

A natural question arises if advertisers will have preference for positions that go beyond the top $\kappa_i$'s. It is possible that there are other considerations that

make lower positions more desirable. For example, the last position may be preferable to the next-last. Also, appearing consistently at the same position may be desirable for some. Some of the advertisers may not seek the topmost positions in order to weed out clickers who do not persist through the topmost advertisements to choose the most appropriate one. Thus, there are a variety of factors that govern the position preference of an advertiser. In the future, this may lead to more general position auctions than the prefix auctions we have studied here. We briefly comment on two variants.

*Arbitrary Ranges.* If we allow top cutoffs (i.e., bidder $i$ can set a valuation $\alpha_i$ and never appear above position $\alpha_i$), we can consider running essentially the same top-down auction: For each position in order from the top, run a simple Vickrey auction (with one winner) among those advertisers whose range includes the position being considered; the winner is allocated the position, pays according to the next-ranked advertiser, and is removed from the pool of advertisers for subsequent auctions.

The difference here is that we can encounter a position $j$ where there are not advertisers willing to take position $j$, but there are still advertisers willing to take positions lower than $j$. (This cannot occur with prefix ranges.) On a typical search page, the search engine must fill in something at position $j$, or else the subsequent positions do not really make sense. Practically speaking, one could fill in this position with some sort of "filler" ad. Given some sort of resolution of this issue, the top-down auction maintains the minimum pay property for general ranges, by essentially the same argument as the prefix case in this paper.

However, the property that there is an equilibrium that matches the VCG outcome is no longer true, as shown by the following example:

*Example 3.* Suppose we have three bidders, and their ranges and valuations are given as follows: `A (1,1) $3`; `B (2,3) $2`; `C (1,3) $1`. We also have three positions, and we get 100, 99 and 98 clicks in them, respectively. The VCG outcome is an allocation of [A,B,C], and prices [\$2, \$1, \$0] (for *all* clicks). To achieve this outcome in the top-down range auction, we must have A with the highest bid, and it is $\infty$ wlog. Since C is the only other bidder competing for the first slot, the price of A (which must be \$2) is determined by the bid of C, and thus $\$2 = p_1 = b_C c_1 = 100 b_C$. Therefore we have that $b_C = 2/100$. Since bidder B wins the second slot, we must have bidder B outbidding bidder C, and the price of B is also determined by the bid of C; so we get $p_2 = b_C c_2 = 99(2/100)$. This is inconsistent with the VCG price of \$1.

*General Position Bids.* One is tempted to generalize the position-based auction so that instead of enforcing a ranking, each advertiser submits separate bids for each position and the market decides which positions are better. Suppose we allowed such bids, and let $b_{i,j}$ denote the bid of advertiser $i$ for position $j$.

In this setting, the ordering property no longer makes sense, but it still might be interesting to consider the minimum-pay property. We do need to clarify our definition of this property; because the advertiser has control of more than just

the bid that gave her the victory; we need to make sure that altering the *other* bids cannot give the advertiser a bargain for this particular position.

A natural mechanism and pricing scheme is as follows: Given the bids $b_{i,j}$, compute the maximum matching of bids to positions (i.e., the VCG allocation). Now for each winning bid $b^i_j$, do the following. delete all other edges from i, and lower this bid until the max matching no longer assigns $i$ to $j$. Set the price per click $\text{ppc}_j$ to the bid where this happens.

Note that this price has the property that if the winner of a position bids between the bid and the price, then they either get the same position at the same price, or perhaps one of their other bids causes them to get a different position. But, we still have the property that the winner cannot get the position she won for a lower price.

It turns out that this is exactly the VCG mechanism, as seen by the following argument. In the following, let $M$ be the valuation of the maximum matching, and for some $i$ let $M_{-i}$ be the valuation of the maximum matching that does not include bidder $i$. Note also that if bidder $i$ is assigned position $j$, the VCG price is $M_{-i} - (M - b_{i,j}c_j)$.

In the suggested auction, when setting the price for bidder $i$, consider the moment when the bid is lowered to $\text{ppc}_j$. The total valuation of the matching at this point is $M - (b_{i,j} - \text{ppc}_j)c_j$. But the valuation of the matching at this point also is equal to $M_{-i}$ since lowering the bid below $\text{ppc}_j$ makes the matching no longer assign $i$ to $j$ (and all other edges are deleted, so $i$ is not assigned anywhere else). So we get $M - (b_{i,j} - \text{ppc}_j)c_j = M_{-i}$, and therefore $\text{ppc}_j c_j = M_{-i} - (M - b_{i,j}c_j)$, which is the VCG price.

## References

1. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. In: Second Workshop on Sponsored Search Auctions. (2006)
2. Varian, H.: Position auctions (2006) Working Paper, available at http://www.sims.berkeley.edu/~hal/Papers/2006/position.pdf.
3. Aggarwal, G.: Privacy Protection and Advertising in a Networked World. PhD thesis, Stanford University (2005)
4. Aggarwal, G., Goel, A., Motwani, R.: Truthful auctions for pricing search keywords. In: ACM Conference on Electronic Commerce (EC06). (2006)
5. Vickrey, W.: Counterspeculation, auctions and competitive sealed tenders. Journal of Finance **16** (1961) 8–37
6. Clarke, E.: Multipart pricing of public goods. Public Choice **11** (1971) 17–33
7. Groves, T.: Incentives in teams. Econometrica **41** (1973) 617–631
8. Nielsen//NetRatings: Interactive advertising bureau (IAB) search branding study (2004) Commissioned by the IAB Search Engine Committee. Available at http://www.iab.net/resources/iab_searchbrand.asp.
9. Mas-Collel, A., Whinston, M., Green, J.: Microeconomic Theory. Oxford University Press (1995)

# Coping with Interference: From Maximum Coverage to Planning Cellular Networks

David Amzallag, Joseph (Seffi) Naor, and Danny Raz

Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel
{amzallag,naor,danny}@cs.technion.ac.il

**Abstract.** Cell planning includes planning a network of base stations providing a coverage of the service area with respect to current and future traffic requirements, available capacities, interference, and the desired quality-of-service. This paper studies cell planning under budget constraints through a very close-to-practice model. This problem generalizes several problems such as budgeted maximum coverage, budgeted unique coverage, and the budgeted version of the facility location problem.

We present the first study of the budgeted cell planning problem. Our model contains capacities, non-uniform demands, and interference that are modeled by a penalty-based mechanism that may reduce the contribution of a base station to a client as a result of simultaneously covering this client by other base stations. We show that this very general problem is **NP**-hard to approximate and thus we define a restrictive version of the problem that covers all interesting practical scenarios. We show that although this variant remains **NP**-hard, it can be approximated within a factor of $\frac{e-1}{2e-1}$ of the optimum.

## 1 Introduction

Consider a set $I = \{1, 2, \ldots, m\}$ of possible configurations of base stations and a set $J = \{1, 2, \ldots, n\}$ of clients. Each base station $i \in I$ has capacity $w_i$, opening cost $c_i$, and every client $j \in J$ has a demand $d_j$. The demand is allowed to be simultaneously satisfied by more than one base station. Each base station $i$ has a *coverage area* represented by a set $S_i \subseteq J$ of clients admissible to be covered (or satisfied) by it; this base station can satisfy at most $w_i$ demand units of the clients in $S_i$.

When a client is belong to the coverage area of more than one base station, interference between the servicing stations may occur. These interference are modeled by a penalty-based mechanism and may reduce the contribution of a base station to a client. Let P be an $m \times m \times n$ matrix of *interference*, where $\mathrm{p}(i_1, i_2, j) \in [0, 1]$ represents the fraction of $i_1$'s service which client $j$ loses as a result of interference with $i_2$ (defining $\mathrm{p}(i, i, j) = 0$ for every $i \in I,\ j \in J$,

T. Erlebach and C. Kaklamanis (Eds.): WAOA 2006, LNCS 4368, pp. 29–42, 2006.

and $p(i, i', j) = 0$ for every $j \notin S_{i'}$)[1]. This means that the interference caused as a result of a coverage of a client by more than one base station depends on the geographical position of the related "client". Followed by the above setting, we denote by $Q(i, j)$ the net contribution of base station $i$ to client $j$, for every $j \in J$, $i \in I$, after incorporating the interference. A detailed description of $Q(i, j)$ is given later in this section.

The *budgeted cell planning problem* (BCPP) asks for a subset of base stations $I' \subseteq I$ whose cost does not exceed a given budget $B$, such that the total number of *fully* satisfied clients is maximized. That is, a solution to BCPP needs to maximize the number of clients for which $\sum_{i \in I} Q(i, j) \geq d_j$.

This problem generalizes several problems such as budgeted maximum coverage [10], budgeted unique coverage [5], and the budgeted version of the facility location problem (analyzed in Section 2.4). So far, these problems were studied (in the sense of approximation algorithms) without considering capacities or non-uniform demands. Coping with interference in covering problems is a great algorithmic challenge; unlike problems where there are no interference, during the time the solution is established, adding a new candidate (e.g., set, bin, item) to the cover may *decrease* the value of the solution. Furthermore, this problem involves full coverage (also known as *all-or-nothing* coverage) which usually makes the approximation task more complex (see [4] for example).

Cell planning is one of the most significant steps in the planning and management of cellular networks and it is among the most fundamental problems in the field of optimization of cellular networks. Cell planning includes planning a network of base stations that provides a (full or partial) coverage of the service area with respect to current and future traffic requirements, available capacities, interference, and the desired QoS. Under these constraints, the objective is, in general, to minimize the operator's total system cost. Cell planning is employed not only when new networks are built or when modifications to a current networks are made, but also (and mainly) when there are changes in the traffic demands, even within a small local area (e.g., building a new mall in the neighborhood or opening new highways). Planning cellular networks under budget limitations is practically the most important optimization problem in the planning stage. Since budget restrictions may lead to failure in achieving the required coverage, the objective, in this case, is hence to maximize the number of covered clients. This paper studies cell planning under budget constraints where the goal is to have a theoretical model that can be used in practical setting.

**Computing $Q(i, j)$.** Our technique for solving BCPP is independent in the structure of $Q(i, j)$. We describe here two general models for computing $Q(i, j)$.

Let $x_{ij}$ be the fraction of the capacity $w_i$ of a base station $i$ that is supplied to client $j$. Recall that $I' \subseteq I$ is the set of base stations selected for opening, the contribution of base station $i$ to client $j$ is, in general is defined by

[1] For simplicity, we do not consider here interference of higher order. These can be further derived and extended from our model.

$$\mathrm{Q}(i,j) = w_i x_{ij} \cdot \prod_{i' \in I'} \big(1 - \mathrm{p}(i,i',j)\big). \tag{1}$$

This means that the net contribution of base station $i$ to client $j$ depends on all other base stations $i'$ that contains $j$ in their coverage areas. Each of these base stations "interferes" base station $i$ to service $j$ and reduces the contribution of $w_i x_{ij}$ by a factor of $\mathrm{p}(i,i',j)$.

Since (1) is a high-order expression we use the following first-order approximation[2]

$$\prod_{i' \in I'} \big(1 - \mathrm{p}(i,i',j)\big) = \big(1 - \mathrm{p}(i,i'_1,j)\big)\big(1 - \mathrm{p}(i,i'_2,j)\big) \ldots \approx 1 - \sum_{i' \in I'} \mathrm{p}(i,i',j) \tag{2}$$

Combining (1) and (2) we get

$$\mathrm{Q}(i,j) \approx \begin{cases} w_i x_{ij}\big(1 - \sum_{i' \in I'} \mathrm{p}(i,i',j)\big), & \sum_{i' \in I'} \mathrm{p}(i,i',j) < 1 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Consider, for example, a client $j$ belonging to the coverage areas of two base stations $i_1$ and $i_2$, and assume that just one of these base stations, say $i_1$, is actually participating in $j$'s satisfaction (i.e., $x_{i_1 j} > 0$ but $x_{i_2 j} = 0$). According to the above model, the mutual interference of $i_2$ on $i_1$'s contribution ($w_1 x_{i_1 j}$) should be considered, although $i_2$ is not involved in the coverage of client $j$.

In most cellular wireless technologies, this is the usual behavior of interference. However, in some cases a base station can affect the coverage of a client *if and only if* it is participating in its demand satisfaction. The contribution of base station $i$ to client $j$ in this case is defined by

$$\mathrm{Q}(i,j) \approx \begin{cases} w_i x_{ij}\Big(1 - \sum_{i' \neq i \in I_j} \mathrm{p}(i,i')\Big), & \sum_{i' \neq i \in I_j} \mathrm{p}(i,i') < 1 \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

where $I_j$ is the set of base stations that participates in the coverage of client $j$, i.e., $I_j = \{i \in I : x_{ij} > 0\}$. Notice that in this model the interference function does not depend on the geographic position of the clients.

**Our contributions.** In this paper we present the first study of the budgeted cell planning problem. To the best of our knowledge, despite the extensive research of non-budgeted cell planning problems (i.e., minimum-cost cell planning, as descried in Section 2.1), there is no explicit study in the literature of the BCPP (in both theoretical and, surprisingly, also in practical settings). We survey, in Section 2, some previous work related to BCPP. Budgeted maximum coverage, budgeted unique coverage, budgeted facility location, and maximizing submodular set functions are among the reviewed problems. In Section 3 we show that approximating BCPP is **NP**-hard. Then we define a restrictive version of BCPP,

[2] Notice that, in this context, one can precisely estimate the "cost" of such approximation using [11,9]. However, for simplicity we do not include these works in this practical model.

the $k4k$-budgeted cell planning, by making additional assumptions that are motivated by practical considerations. The additional property is that every set of $k$-opened base stations can fully satisfy at least $k$ clients, for every integral value of $k$. In Section 4 we show that this problem remains **NP**-hard and present an $\frac{e-1}{2e-1}$ ($\approx 0.3873$) factor approximation algorithm for this problem.

## 2 Related Work

### 2.1 The Minimum-Cost Cell Planning Problem

The *minimum-cost cell planning problem* asks for a minimum-cost subset $I' \subseteq I$ that satisfies the demands of *all* the clients. This important problem is one of the most studied on the area of cellular network optimization. Previous work dealt with a wide variety of special cases (e.g., cell planning without interference, frequency planning, uncapacitated models, antenna-type limitations, and topological assumptions regarding coverage). These works range from meta-heuristics (e.g., genetic algorithms, simulated annealing, etc.) and greedy approaches, through exponential-time algorithms that compute an optimal solution, to approximation algorithms for special cases of the problem. A comprehensive survey of various works on minimum-cost cell planning problems appears in [3]. An $O(\log W)$-approximation algorithm for the non-interference version of the minimum-cost cell planning problem is presented in [2], where $W$ is the largest given capacity of a base station.

### 2.2 Base Stations Positioning Under Geometric Restrictions

A PTAS for the uncapacitated BCPP with unit demands (i.e., $w_i = \infty$ and $d_j = 1$ for all $i \in I, j \in J$) and without interference is given in [7]. In this case the problem is studied under geometric restrictions of disks of constant radius $D$ (i.e., $S_i$ is the set of clients located within a distance no greater than $D$ from the geometric location of $i$, for every $i \in I$), a minimal distance between different base stations that have to be kept, and clients as well as base stations are associated with points in the Euclidean plane.

### 2.3 Budgeted Maximum Coverage and Budgeted Unique Coverage

BCPP is closely related to the budgeted maximum coverage and the budgeted unique coverage version of set cover. Given a collection of subsets $\mathcal{S}$ of a universe $U$, where each element in $U$ has a specified weight and each subset has a specified cost, and a budget $B$. The *budgeted maximum coverage problem* asks for a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ of sets, whose total cost is at most $B$, such that the total weight of elements covered by $\mathcal{S}'$ is maximized. The *budgeted unique coverage problem* is a similar problem where elements in the universe are uniquely covered, i.e., appears in exactly one set of $S'$. Both problems are special cases of BCPP in which elements are clients with unit demands, every set $i \in I$ corresponds to a base station $i$ containing all clients in its coverage area $S_i \subseteq J$, and $w_i \geq |S_i|$ for

all base stations in $I$. In this setting, budgeted maximum coverage is the case (in the sense that a solution for BCPP is optimal if and only if it is optimal for the budgeted maximum coverage) when there are no interference (i.e., P is the zero matrix), while budgeted unique coverage is when the interference is taking to be the highest (i.e., $\mathrm{p}(i', i'', j) = 1$ for every $i' \neq i''$, and $\mathrm{p}(i', i'', j) = 0$ otherwise).

For the budgeted maximum coverage problem, there is a $(1-\frac{1}{e})$-approximation algorithm [10,1], and this is the best approximation ratio possible unless **NP=P** [6]. For the budgeted unique coverage problem, there is an $\Omega(1/\log n)$-approximation algorithm [5] and, up to a constant exponent depending on $\epsilon$, $O(1/\log n)$ is the best possible ratio assuming $\mathbf{NP} \nsubseteq \mathbf{BPTIME}(2^{n^\epsilon})$ for some $\epsilon > 0$. Interestingly enough, we will show in the next section that our generalization for both of these problems is hard to approximate.

### 2.4 Budgeted Facility Location

The budgeted version of the (uncapacitated) facility location problem is also closely related to BCPP. In the traditional (uncapacitated) facility location problem we wish to find optimal locations in which to build facilities, from a given set $I$, to serve a given set $J$ of clients, where building a facility in location $i$ incurs a cost of $f_i$. Each client $j$ must be assigned to one facility, thereby incurring a cost of $c_{ij}$ (without assuming the triangle inequality). The objective is to find a solution of minimum total cost. The *budgeted facility location problem* is to find a subset $I' \subseteq I$ such that the total cost of opening facilities and connecting clients to open facilities does not exceed a given budget $B$, and the total number of connected clients is maximized.

Given an instance of the budgeted (uncapacitated) facility location problem, we show in the following that this problem is a special case of the budgeted maximum coverage problem. By a *star* we mean a pair $(i, Q)$ with $i \in I$ and $Q \subseteq J$. The cost of a star $(i, Q)$ is $c(i, Q) = f_i + \sum_{j \in Q} c_{ij}$, and its *effectiveness* is $\frac{|Q|}{c(i,Q)}$. Then the budgeted (uncapacitated) facility location problem is a special case of the budgeted maximum coverage problem: set $J$ is the set of elements that need to be covered, and let $S = 2^J$, where $c(Q)$ is the minimum-cost of a star $(i, Q)$ (we take the same budget for both instances).

However, the resulting budgeted maximum coverage instance has exponential size, and therefore this reduction cannot be used directly. Nevertheless, we can apply the algorithm of [10] without generating the instance explicitly, as proposed by Hochbaum [8]: In each step, we have to find a most effective star, open its facility and henceforth disregard all clients in this star. Although there are exponentially many stars, it is easy to find the most effective one as it suffices to consider stars $(i, Q_k^i)$, for $i \in I$ and $k \in \{1, 2, \ldots, |J|\}$. Here $Q_k^i$ denotes the first $k$ clients in a linear order with nondecreasing $c_{ij}$. Clearly, other stars cannot be more effective. Hence, we get the same approximation ratio as the budgeted maximum coverage problem. Moreover, since the budgeted maximum coverage can be described, by a simple reduction, as a special case of the budgeted facility location, the best we can hope for the budgeted facility location problem is the same approximation factor as the budgeted maximum coverage problem.

### 2.5 Maximizing Submodular Set Functions

Let $U = \{1, \ldots, n\}$, let $c_u, u \in U$ be a set of nonnegative weights, and let $B$ be a nonnegative budget. The problem of *maximizing nondecreasing submodular set function with budget constraint* is

$$\max_{S \subseteq U} \left\{ f(S) : \sum_{u \in S} c_u \leq B \right\},$$

where $f(S)$ is a nonnegative nondecreasing submodular polynomially computable set function (a set function is submodular if $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$ for all $S, T \subseteq U$ and nondecreasing if $f(S) \leq f(T)$ for all $S \subseteq T$). For this problem, there is a $(1 - \frac{1}{e})$-approximation algorithm [12], and as this problem is a generalization of the budgeted maximum coverage ($c_u = 1$, for all $u \in U$, and $f(S)$ denotes the maximum weight that can be covered by the set $S$), this ratio is the best achievable.

Although this problem seems, at least from a natural perspective, to be closely related to BCPP, observe that set (covering) functions are not submodular, in general, when interference are involved. Consider, for example, an instance of BCPP in which $I = \{1, 2, 3\}$ with $w_1 = w_2 = 1$ and $w_3 = 1/4$, a single client with $d = 2$ that can be satisfied by all base stations, and symmetric penalties $\mathrm{p}(1, 3) = \mathrm{p}(2, 3) = 1/2$, while $\mathrm{p}(1, 2) = 0$. Taking $S = \{1\} \cup \{3\}$ and $T = \{2\} \cup \{3\}$ we have $f(S) + f(T) \ngeq f(S \cup T) + f(S \cap T)$, where $f(S)$ is defined to be the maximum number of fully satisfied clients that can be covered by the set $S$ of base stations.

## 3 Inapproximability

As mentioned earlier, the budgeted maximum coverage as well as budgeted unique coverage can be seen as special cases of BCPP. In both cases the approximation algorithms are based on the greedy technique of Khuller, Moss, and Naor [10]. This means picking at each step the most effective set until either no element is left to be covered or the budget limitation is exceeded. Combining this method with the enumeration technique yields the $(1 - \frac{1}{e})$-approximation algorithm of [10].

Unfortunately, a natural attempt to adapt the ideas from [10] to the setting of BCPP fails, as stated by the next theorem.

**Theorem 1.** *It is* **NP***-hard to find a feasible solution to the budgeted cell planning problem.*

**Proof.** The proof is via a reduction from the subset sum problem. Given an instance of the subset sum problem, i.e., a set of natural numbers $A = \{a_1, a_2, \ldots, a_n\}$ and an additional natural number $T = \frac{1}{2} \sum_{i=1}^{n} a_i$. We build an instance of the budgeted BCPP with $I = \{1, 2, \ldots, n\}, |J| = 1$ and $w_i = c_i = a_i$ for every $i \in I$; the budget and the single client's demand are $B = d = T$ and no interference are assumed.

It is easy to see that the client is satisfied if and only if there exists $S \subseteq A$ with $\sum_{i \in S} a_i = T$. Since there is only a single client, any polynomial-time approximation algorithm must produce a full coverage, solving the subset sum problem in polynomial time. ■

## 4 The *k*4*k*-Budgeted Cell Planning Problem

In light of the above inapproximability result, we turn to define a restrictive version of BCPP which is general enough to cover all interesting practical cases. In order to do that, we use the fact that in general, the number of base stations in cellular networks is much smaller than the number of clients. Notice that when planning cellular networks, the notion of "clients" sometimes means mobile-clients and sometimes it represents the total traffic demand created by many mobile-clients at a given location. Our models support both forms of representations. Moreover, when there is a relatively large cluster of antennas in a given location, this cluster is usually addressed to meet the traffic requirements of a high-density area of clients. Thus for both interpretations of "clients" the number of satisfied clients is always much bigger than the number of base stations. Followed by the above discussion, we define the *k*4*k*-*budgeted cell planning problem* (*k*4*k*-BCPP) to be BCPP with the additional property that every set of $k$ base stations can fully satisfy at least $k$ clients, for every integer $k$ (and we refer to this property as "*k*4*k* property").

In this section we show that this problem can be approximated within a factor of $\frac{e-1}{2e-1}$ of the optimum. First, we show that this problem remains hard.

**Theorem 2.** *The k*4*k-budgeted cell planning problem is* **NP***-hard.*

**Proof.** Via a reduction from the budgeted maximum coverage problem. Consider an instance of the budgeted maximum coverage problem, that is, a collection of subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$ with associated costs $\{c_i\}_{i=1}^m$ over a domain of elements $X = \{x_1, \ldots, x_n\}$, and a budget $L$.

We can construct an instance of *k*4*k*-BCPP such that an optimal solution to this problem gives an optimal solution to the budgeted maximum coverage problem. First, we construct a bipartite graph of elements vs. sets, derived from the budgeted maximum coverage instance: there is an edge $(x_i, S_j)$ if and only if element $x_i$ belongs to set $S_j$. The instance of *k*4*k*-BCPP is as follows: the set of clients is $\{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_m\}$, where each of the $x_j$'s is of unit demand and each of the $y_r$'s is of zero demand, the set of potential base stations is $\{S_1, \ldots, S_m\}$, each of opening cost $c_i$, a capacity $w_i = |S_i|$, and a set of admissible clients for covering $S_i \cup \{y_1, \ldots, y_m\}$, for every $i = 1, \ldots, m$, and $j = 1, \ldots, n$, and a budget $B = L$, while no interference are assumed.

Clearly, a solution to *k*4*k*-BCPP is optimal if and only if the corresponding solution of the budgeted maximum coverage instance is optimal. ■

### 4.1 The Structure of BCPP Solutions

Our algorithm is based on a combinatorial characterization of the solution set to BCPP[3] (and in particular to $k4k$-BCPP ). The following lemma is a key component in the analysis of our approximation algorithm.

**Lemma 1.** *Every solution to the $k4k$-budgeted cell planning problem can be transformed to a solution in which the number of clients that are covered by more than one base station is at most the number of opened base stations. Moreover, this transformation leaves the number of fully satisfied clients as well as the solution cost unchanged.*

**Proof.** Consider a solution $\Delta = \{I', J', \mathbf{x}\}$ to the $k4k$-BCPP , where $I' \subseteq I$ is the set of base stations selected for opening, $J' \subseteq J$ is the set of fully satisfied clients, $x_{ij}$'s are the base station-client coverage rates, and $J'' \subseteq J'$ is the set of clients that are satisfied by more than one base station. Without loss of generality we may assume that every client has a demand greater than zero, since there is no need for "covering" clients with zero demand. We associate the weighted bipartite graph $G_\Delta = (I' \cup J', E)$ with every such solution. In this graph, $(i, j) \in E$ has weight $w(i, j) = w_i x_{ij}$ if and only if $x_{ij} > 0$, and $w(i, j) = 0$, otherwise. Two cases need to be considered:

1. If $G_\Delta$ is acyclic then we are done (i.e., no transformation is needed); in this case $|J''| < |I'|$. To see this, let $T$ be a forest obtained from $G_\Delta$ by fixing an arbitrary base station vertex as the root (in each of the connected components of $G_\Delta$) and trimming all client leaves. These leaves correspond to clients who are covered, in the solution, by a single base station. Since the height of the tree is even, the number of internal client-vertices is at most the number of base station-vertices, hence $|J''| < |I'|$.
2. Otherwise, we transform $G_\Delta = (I' \cup J', E)$ into an acyclic bipartite graph $G_{\Delta'} = (I' \cup J', E')$ using a cycle canceling algorithm. For simplicity, we first describe the following algorithm for the case that interference do not exist.

   > **Algorithm 1** [CYCLE CANCELING WITHOUT INTERFERENCE]**.** As long as there are cycles in $G_\Delta$, pick a cycle $C$ and let $\gamma$ be the weight of a minimum-weight edge on this cycle. Take a minimum-weight edge on $C$ and, starting from this edge, alternately, in clockwise order along the cycle, decrease and increase the weight of every edge by $\gamma$.

   It is easy to verify that at the end of the algorithm every client receives, and every base station supplies, the same amount of demand units as before. Moreover, the only changes here are the values of the $x_{ij}$'s. Hence, Algorithm 1 preserves the number as well as the identity of the satisfied clients. Since at each iteration at least one edge is removed, $G_{\Delta'}$ is acyclic, thus yielding $|J''| < |I'|$ as in the former case.

[3] Results in this section can be applied also for non-$k4k$ versions of the BCPP. For simplicity, we concentrate here on the $k4k$ versions of the problem.

When interferences exist but $\mathrm{Q}(i,j)$ is independent on the $x_{ij}$'s, we can still use Algorithm 1 to preserves the number as well as the identity of the satisfied clients. In this case change in the $x_{ij}$'s does not affect the $\mathrm{Q}(i,j)$ of any client. However, when $\mathrm{Q}(i,j)$ is a function of the $x_{ij}$'s this algorithm can no longer guarantee this. This is true because of the way the fully satisfied clients "use" base stations not on the cycle depends on the interference, and thus the modifications of the edge weights on the cycle are not enough. To overcome this problem we generalize the method of cycle canceling. Consider a cycle $C = (v_1, \ldots, v_k = v_1)$ in $G_\Delta$, such that odd vertices correspond to base stations. Let $v_i$ be any client-vertex in $C$. Now suppose the base station which corresponds to $v_{i-1}$ increases its supply to $v_i$ by $\alpha$ units of demand. The basic idea of the generalization is to compute the exact number of demand units the base station which corresponds to $v_{i+1}$ *must* subtract from its coverage, in order to preserve the satisfaction of that client, taking into account all the demand (with its interference) supplied by base station vertices which are outside the cycle.

Notice that increasing a certain $w(v_i, v_{i+1})$ *does not necessary* increase the supply to client $v_i$. When interferences are considered, it could actually happen that increasing $w(v_i, v_{i+1})$ *decreases* the supply to $v_i$ (if the new interference penalties outweigh the increased supply). Similarly, decreasing some $w(v_i, v_{i+1})$ could actually increase the supply to $v_i$. However, one can assume for optimal solutions that these cases do not occur (as the solution could be transformed into an equivalent solution where such edges have $w(v_i, v_{i+1}) = 0$).

To demonstrate the idea of canceling cycles when interferences exist let us assume, for simplicity, that there is only a single base station which is not on the cycle, denoted by $v_o$, which participates in the coverage of client $v_i$, and the interference model is assumed to be the one in (4). Then, the total contribution of base stations $v_{i-1}, v_{i+1}$, and $v_o$ to the coverage of client $v_i$ is, by (1),

$$\delta(v_i) = \mathrm{Q}(v_0, v_i) + \mathrm{Q}(v_{i+1}, v_i) + \mathrm{Q}(v_{i-1}, v_i).$$

Given that the supply of base station $v_{i-1}$ to client $v_i$ is increased by $\alpha$ units of demand (i.e., $w'(v_{i-1}, v_i) = w(v_{i-1}, v_i) + \alpha$, where $w'$ is the updated weight function of the edges), base station $v_{i+1}$ must decrease its supply to this client by $\beta$ units of demand (i.e., $w'(v_{i+1}, v_i) = w(v_{i+1}, v_i) - \beta$) in order to preserve the satisfaction of client $v_i$ (assuming $v_o$'s supply remains the same). Then, the value of $\beta$ can be computed via a solution to the following equation (in variable $\beta$),

$$\delta'(v_i) = \mathrm{Q}'(v_0, v_i) + \mathrm{Q}'(v_{i+1}, v_i) + \mathrm{Q}'(v_{i-1}, v_i) = \delta(v_i).$$

Notice that our cycle canceling algorithms are used for the proof of existence and such computations are not necessary for the execution of our approximation algorithm.

**Algorithm 2** [CYCLE CANCELING WITH INTERFERENCE]**.** As long as there are cycles in $G_\Delta$, pick a cycle $C = (v_1, \ldots, v_k = v_1)$ where odd vertices represent base stations. As before, every edge $e_i = (v_i, v_{i+1})$ on the cycle has a weight $w(v_i, v_{i+1})$ associated with it, representing the amount of demand supplied by the base-station-vertex in $e_i$ to the client-vertex in $e_i$. For simplicity, let $d'_i$ denote this value.

We recursively define a sequence of weights $\{y_i\}_{i=1}^{k-1}$, with alternating signs which represent a shift in the demand supply of base stations to clients along the cycle. Start by setting $y_1 = \epsilon$, representing an increase of $\epsilon$ to the demand supplied by the base-station-vertex $v_1$ to the client-vertex $v_2$. This increase of supply may not all be credited to vertex $v_2$ due to interference, some of which are possibly due to base stations which are outside the cycle, which contribute to $v_2$'s satisfaction. Set $y_2$ to be the maximal decrease in the demand supplied by base-station-vertex $v_3$ to client-vertex $v_2$, such that $v_2$ remains satisfied. This amount of demand is now "available" to base-station-vertex $v_3$, hence we allow $v_3$ to supply this surplus to client-vertex $v_4$. We continue in this manner along the cycle.

If, by repeating this procedure, we end up with $|y_{k-1}| \geq \epsilon$, then we say the cycle is $\epsilon$-*adjustable*. Otherwise, redefine the values of $\{y_i\}_{i=1}^{k-1}$ in a similar manner, but in reverse order, i.e., starting from $y_{k-1}$ and ending with $y_1$. However, it is easy to verify that at least one direction, the cycle is $\epsilon$-adjustable, for some value of $\epsilon$.

Let $\epsilon_{\max}$ be the largest value for which the cycle is adjustable, and consider its corresponding values of $y_i$, $i = 1, \ldots, k-1$. Note that the $y_i$'s have alternating signs, and for any client-vertex $v_i$, $y_i = -y_{i-1}$. Define the quotients $z_i = d'_i/y_i$ for every $i = 1, \ldots, k-1$, and let $z_{\min} = \min_{z_i<0} |z_i|$. Now increase the amount of demand supplied on every edge on the cycle to be $w'(v_i, v_{i+1}) = y_i \cdot z_{\min}$, where $w'$ is the updated weight function of the edges, as before.

Two important invariants are maintained throughout our cycle-canceling procedure. The first is that $w'(i,j) \geq 0$ for every edge $(i,j)$ of the cycle. The second is that there exists at least one edge $e = (i,j)$ on the cycle for which $w'(i,j) = 0$. Therefore Algorithm 2 preserves the number and the identity of the satisfied clients and $G_{\Delta'}$ is also a solution. Since at each iteration at least one edge is removed, $G_{\Delta'}$ is acyclic and $|J''| < |I'|$ as before. ■

### 4.2 An $\frac{e-1}{2e-1}$-Approximation Algorithm

We are now ready to present a $\frac{e-1}{2e-1}$-approximation algorithm for $k4k$-BCPP. We combine ideas from [10] together with our characterization of the optimal solution set of $k4k$-BCPP. Throughout this section we use the following notation. Let $N_i$ be the maximum number of clients that can be covered by a single base station $i$ (i.e., without allowing simultaneously covering of a client), $i = 1, 2, \ldots, m$. Let $N(I')$ denote the total number of clients that can be covered by $I'$ in such a way that each client is covered by a single base station, and let

**Algorithm 3.** $k4k$-BUDGETED CELL PLANNING

$J' \leftarrow \emptyset$; $H_3 \leftarrow 0$;
$H_1 \leftarrow$ maximum number of base stations having a total opening cost less than or equal to $B$
$H_2 \leftarrow \text{argmax}\{N(S),$ such that $S \subseteq I$, $|S| < \ell$, and $c(S) \leq B\}$
**for all** $S \subseteq I$, such that $|S| = \ell$ and $c(S) \leq B$ **do**
    $\mathcal{I} \leftarrow I \setminus S$
    **repeat**
        select $i \in \mathcal{I}$ that maximizes $\frac{N'_i}{c_i}$
        **if** $c(S) + c_i \leq B$ **then**
            $S \leftarrow S \cup \{i\}$
            $J' \leftarrow J' \cup J_i$
            update $w_i$ by the demand units supplied to $J_i$
            $c(S) \leftarrow c(S) + c_i$
        **end if**
        $\mathcal{I} \leftarrow \mathcal{I} \setminus \{i\}$
    **until** $\mathcal{I} = \emptyset$
    **if** $N(S) > H_3$ **then** $H_3 \leftarrow N(S)$
**end for**
Output the solution having the largest value from $\{H_1, H_2, H_3\}$

$N'_i$ denote the maximum number of clients that can be covered by a single base station $i$, but not covered by any other base stations in $I'$. Finally, we denote by $J_i$ the set of clients that are fully satisfied by base station $i$. Without loss of generality we may assume that the opening cost of any base station does not exceed $B$, since base stations of cost greater than $B$ do not belong to any feasible solution.

We first observe that the greedy algorithm that opens at each step a base station maximizing the ratio $\frac{N'_i}{c_i}$ has an unbounded approximation factor. Consider, for example, two base stations and $M+2$ clients $J = \{1, \ldots, M, M+1, M+2\}$ having unit demands. Let $w_1 = 2, c_1 = 1, S_1 = \{M+1, M+2\}$, where $w_2 = c_2 = M, S_2 = \{1, \ldots, M\}$. The overall budget in this example is taken to be $M$. The optimal solution opens the second base station satisfying exactly $M$ clients, while the solution obtained by the greedy algorithm opens the first base station satisfying exactly 2 clients. The approximation ratio for this instance is $M/2$, and is therefore unbounded.

Our algorithm comprises of two phases. In the first phase, the algorithm computes the maximum number of base stations having a total opening cost less than or equal to $B$. Since our instances are "$k4k$", this is a lower bound on the optimal solution of $k4k$-BCPP . Furthermore, it can be computed in linear-time by picking base stations in non-decreasing order of their opening cost. Another set of candidate solutions concentrates, in the second phase, on the number of clients that can be fully satisfied by a single base station. This is also a lower bound on the optimal solution and it is computed as the best of two possible candidates (in a similar way to [10]). For a fixed integer $\ell \geq 3$, the first candidate

consists of all subsets of $I$ of cardinality less than $\ell$ which have cost at most $B$, while the second one enumerates all feasible solutions of cardinality $\ell$ having cost at most $B$, and then completes each subset to a candidate solution using the greedy algorithm. Based on both phases the algorithm outputs the candidate solution having the maximum number of satisfied clients.

The problem of computing the optimal value of $N(S)$, for a given set of base stations, $S$, is **NP**-hard. In fact, this problem is a generalization of the budgeted maximum coverage problem containing capacities as well as non-uniform demands. Fortunately, a straightforward extension of [10] gives a $(1-\frac{1}{e})$-approximation algorithm for this generalization.

**Theorem 3.** *Algorithm 3 is a* $\frac{e-1}{2e-1}$*-approximation algorithm for the k4k-budgeted cell planning problem.*

**Proof.** Let $\tilde{n}$ be the solution obtained by Algorithm 3, and let $n^*$ be the maximum number of satisfied clients as obtained by the optimal solution. In the latter, $n_1^*$ denotes the number of clients that are satisfied by a single base station, and $n_2^*$ is the number of clients satisfied by more than one base station. Finally, we denotes $I^*$ to be the set of base stations opened (by the optimal solution) for satisfying these $n^* = n_1^* + n_2^*$ clients.

Now, if $\hat{n}_1$ denotes the maximum number of clients that can be satisfied by a single base station then $\hat{n}_1 \geq n_1^*$. Since computing $\hat{n}_1$ is done using the extension of the algorithm of [10], we have

$$\tilde{n} \geq \left(1 - \frac{1}{e}\right)\hat{n}_1. \tag{5}$$

Combining the above discussion, gives

$$\begin{aligned}
\left(2 - \frac{1}{e}\right)\tilde{n} &= \tilde{n} + \left(1 - \frac{1}{e}\right)\tilde{n} && (6)\\
&\geq \tilde{n} + \left(1 - \frac{1}{e}\right)|I^*| && (7)\\
&\geq \left(1 - \frac{1}{e}\right)\hat{n}_1 + \left(1 - \frac{1}{e}\right)|I^*| && (8)\\
&\geq \left(1 - \frac{1}{e}\right)n_1^* + \left(1 - \frac{1}{e}\right)n_2^* && (9)\\
&\geq \left(1 - \frac{1}{e}\right)n^* && (10)
\end{aligned}$$

where inequality (7) follows from the fact that every set of $k$ opened base stations can satisfy at least $k$ clients (as used by the first candidate of our algorithm), inequality (8) is based on (5), and inequality (9) follows from Lemma 1. ■

**Finding $x_{ij}$'s values.** Algorithm 3 outputs a set $J' \subseteq J$ of fully satisfied clients and a set $I' \subseteq I$ of base stations providing this coverage. However, the values

of the $x_{ij}$'s are not inclusively outcome from the algorithm. Since in this setting we are given a set of *already opened* base stations, these values can be efficiently determined by any feasible solution of the following linear program (**LP**). Notice that the objective function in this linear program is not important and any feasible point will do.

$$\textbf{max} \sum_{i \in I'} \sum_{j \in J'} x_{ij} \qquad \textbf{(LP)}$$

$$\textbf{s.t.} \sum_{i \in I'} \mathrm{Q}(i,j) \geq d_j \qquad \forall\, j \in J' \qquad (11)$$

$$\sum_{j \in J'} x_{ij} \leq 1 \qquad \forall\, i \in I' \qquad (12)$$

$$0 \leq x_{ij} \leq 1 \qquad \forall\, i \in I', j \in S_i$$

$$x_{ij} = 0 \qquad \forall\, i \in I', j \notin S_i$$

In this linear program constraints (11) ensures that every client will be fully satisfied (notice that $\mathrm{Q}(i,j)$ is the same as in (3) without the need to open base stations), while constraints (12) maintains the capacity bounds for the base stations.

## 5 Conclusions and Future Work

In this paper we present a theoretical study of the budgeted cell planning, a central complex optimization problem in planning of cellular networks. As far as we know, no performance guarantee was given so far to this problem. We show that although this problem is **NP**-hard to approximate, we can still cover all practical scenarios by adopting a very practical assumption, called the $k4k$-property, satisfied by every real cellular network, and we give a fully combinatorial $\frac{e-1}{2e-1}$-approximation algorithm for this problem. We believe that taking capacities, non-uniform demands, and interference into considerations makes a significant step towards making approximation algorithms a key ingredient in practical solutions to many planning and covering problems in cellular networks.

An interesting open problem that is closely related to BCPP is the all-or-nothing demand maximization problem. In this problem we are given a set $I = \{1, 2, \ldots, m\}$ of base stations that are already opened, a set $J = \{1, 2, \ldots, n\}$ of clients. Each base station $i \in I$ has capacity $w_i$, and every client $j \in J$ has a profit $p_j$ and a demand $d_j$ which is allowed to be simultaneously satisfied by more than one base station. Each base station $i$ has a *coverage area* represented by a set $S_i \subseteq J$ of clients admissible to be covered (or satisfied) by it. Let P be an $m \times m \times n$ matrix of *interference* for satisfying a client by several base stations, as in BCPP. The *all-or-nothing demands maximization problem* asks for a maximum-profit subset $J' \subseteq J$ of clients that can be fully satisfied by $I$. As one can noticed, this problem is a special case of BCPP by taking $c_i = 0$, and $p_j = 1$, for every $i \in I, j \in J$.

## Acknowledgments

We would like to thank Gabi Scalosub for his comments for an earlier version of this paper. This research was supported by REMON - Israel 4G Mobile Consortium, sponsored by Magnet Program of the Chief Scientist Office in the Ministry of Industry and Trade of Israel.

## References

1. A. Ageev and M. Sviridenko. Approximation algorithms for maximum coverage and max cut with given sizes of parts. In *Proceedings of the Conference on Integer Programming and Combinatorial Optimization*, volume 1610 of *Lecture Notes in Computer Science*, pages 17–30. Springer-Verlag, Berlin, 1999.
2. D. Amzallag, R. Engelberg, J. Naor, and D. Raz. Approximation algorithms for cell planning problems. Manuscript, 2006.
3. D. Amzallag, M. Livschitz, J. Naor, and D. Raz. Cell planning of 4G cellular networks: Algorithmic techniques, and results. In *Proceedings of the 6th IEE International Conference on 3G & Beyond (3G'2005)*, pages 501–506, 2005.
4. C. Chekuri, S. Khanna, and F. B. Shepherd. The all-or-nothing multicommodity flow problem. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 156–165, 2004.
5. E. D. Demaine, U. Feige, M. Hajiaghayi, and M. R. Salavatipour. Combination can be hard: Approximability of the unique coverage problem. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 162–171, 2006.
6. U. Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45:634–652, 1998.
7. C. Glaßer, S. Reith, and H. Vollmer. The complexity of base station positiong in cellular networks. In *Workshop on Approximation and Randomized Algorithms in Communication Networks*, pages 167–177, 2000.
8. D. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22(2):148–162, 1982.
9. J. Kahn, N. Linial, and A. Samorodnitsky. Inclusion-exclusion : exact and approximate. *Combinatorica*, 16:465–477, 1996.
10. S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70:39–45, 1999.
11. N. Linial and N. Nisan. Approximate inclusion-exclusion. *Combinatorica*, 10:349–365, 1990.
12. M. Sviridenko. A note on maximizing a submodular set function subject to knapsack constraint. *Operations Research Letters*, 32:41–43, 2004.

# Online Dynamic Programming Speedups*

Amotz Bar-Noy[1], Mordecai J. Golin[2], and Yan Zhang[2]

[1] Brooklyn College, 2900 Bedford Avenue Brooklyn, NY 11210
amotz@sci.brooklyn.cuny.edu
[2] Hong Kong University of Science and Technology, Kowloon, Hong Kong
{golin,cszy}@cse.ust.hk

**Abstract.** Consider the Dynamic Program $h(n) = \min_{1 \le j \le n} a(n,j)$ for $n = 1, 2, \ldots, N$. For arbitrary values of $a(n,j)$, calculating all the $h(n)$ requires $\Theta(N^2)$ time. It is well known that, if the $a(n,j)$ satisfy the *Monge property*, then there are techniques to reduce the time down to $O(N)$. This speedup is inherently static, i.e., it requires $N$ to be known in advance.

In this paper we show that if the $a(n,j)$ satisfy a stronger condition, then it is possible, without knowing $N$ in advance, to compute the values of $h(n)$ in the order of $n = 1, 2, \ldots, N$, in $O(1)$ amortized time per $h(n)$. This *maintains the DP speedup online*, in the sense that the time to compute all $h(n)$ is $O(N)$. A slight modification of our algorithm restricts the worst case time to be $O(\log N)$ per $h(n)$, while maintaining the amortized time bound. For $a(n,j)$ that satisfy our stronger condition, our algorithm is also simpler to implement than the standard Monge speedup.

We illustrate the use of our algorithm on two examples from the literature. The first shows how to make the speedup of the $D$-median on a line problem in an online settings. The second shows how to improve the running time for a DP used to reduce the amount of bandwidth needed when paging mobile wireless users.

## 1 Introduction

Consider the class of problems defined by

$$h(n) = \min_{1 \le j \le n} a(n,j), \quad \forall\ 1 \le n \le N \tag{1}$$

where the goal is to compute $h(n)$ for $1 \le n \le N$. In many applications, (1) is a Dynamic Program (DP), in the sense that the values of $a(n,j)$ depend upon $h(i)$, for some $1 \le i < n$. In this paper, we always assume any particular $a(n,j)$ can be computed in $O(1)$ time, provided that the values of $h(i)$ it depends on are known. For a generally defined function $a(n,j)$, it requires $\Theta(N^2)$ time to compute all the $h(n)$. It is well known, though [1], that if the values of $a(n,j)$ satisfy the *Monge property* (see Section 1.1), then the SMAWK algorithm [2] can compute all the $h(n)$, for $1 \le n \le N$, in $O(N)$ time. To be precise, if

* The research of the second and third authors was partially supported by Hong Kong RGC CERG grant HKUST6312/04E.

T. Erlebach and C. Kaklamanis (Eds.): WAOA 2006, LNCS 4368, pp. 43–54, 2006.

1. the value of $N$ is known in advance;
2. and for any $1 \leq j \leq n \leq N$, the value of $a(n, j)$ can be computed in $O(1)$ time, i.e., $a(n, j)$ does not depend on $h(i)$;
3. and the values of $a(n, j)$ satisfy the Monge property defined by (4),

then the SMAWK algorithm [2] can compute all of the $h(n)$ for $1 \leq n \leq N$ in $O(N)$ time.

The main purpose of this paper is to consider the DP formula (1) in *online* settings. By this we mean that the values of $h(n)$ are computed in the order $n = 1, 2, \ldots, N$ *without* knowing the parameter $N$ in advance, and the values of $a(n, j)$ are allowed to depend on *all* previously-computed values of $h(i)$ for $1 \leq i < n$. To be precise, our main result is

**Theorem 1.** *Consider the DP defined by (1). If*

1. $\forall\ 1 \leq j \leq n \leq N$, *the value of* $a(n, j)$ *can be computed in* $O(1)$ *time, provided that the values of* $h(i)$ *for* $1 \leq i < n$ *are known;*
2. *and* $\forall\ 1 \leq j < n \leq N$,

$$a(n, j) - a(n-1, j) = c_n + \delta_j \beta_n \tag{2}$$

*where* $c_n$, $\beta_n$ *and* $\delta_j$ *are constants satisfying*
*(a)* $\forall\ 1 < n \leq N$, $\beta_n \geq 0$;
*(b) and* $\delta_1 > \delta_2 > \cdots > \delta_{N-1}$,

*then, there is an algorithm that computes the values of* $h(n)$ *in the order* $n = 1, 2, \ldots, N$ *in* $O(1)$ *amortized and* $O(\log N)$ *worst-case time per* $h(n)$. *The algorithm does not know the value of* $N$ *until* $h(N)$ *has been computed.*

We call the Condition 2 in Theorem 1 (including Conditions (a) and (b)) the *online Monge property*. As we will see in Section 1.1, the online Monge property is a stronger Monge property. The SMAWK algorithm is a $\Theta(N)$ speedup of the computation of (1) when $a(n, j)$ satisfy the Monge property. Theorem 1 says that if $a(n, j)$ satisfy the online Monge property, then the same speedup can be maintained online, in the sense that the time to compute all $h(n)$ is still $O(N)$. Section 2 will give the main algorithm, which achieves the $O(1)$ amortized bound. In Section 2.3, we modify the algorithm a little bit to achieve the worst case $O(\log N)$ bound. Section 3 shows two applications of this technique.

Note that the online Monge property only says that $c_n$, $\beta_n$ and $\delta_j$ exist. It does not say that $c_n$, $\beta_n$ and $\delta_j$ are given. However, if $\delta_j$ is given, then the algorithm will be easier to understand. So, throughout this paper we will assume we have an extra condition:

– The values of $\delta_j$ can be computed in $O(1)$ time, provided that the values of $h(i)$ for $1 \leq i < j$ are known.

This condition is not really necessary. In Appendix A, we will show how it is implied by other conditions in Theorem 1.

As a final note we point out that there is a body of literature already discussing "online" problems of (1), e.g., [3,4,5,6,7]. We should clarify that the "online" in

those papers actually had a different meaning than the one used here. More specifically, the result they have is that if

1. the value of $N$ is known in advance;
2. and for any $1 \leq j \leq n \leq N$, the value of $a(n, j)$ can be computed in $O(1)$ time, provided that the values of $h(i)$ for $1 \leq i < j$ are known;
3. and the values of $a(n, j)$ satisfy the Monge property defined by (4),

then both the Galil-Park algorithm [6] and the Larmore-Schieber algorithm [7] can compute all of the $h(n)$ for $1 \leq n \leq N$ in $O(N)$ time. As we can see, their definition of "online" is only that the $a(n, j)$ can depend upon part of the previously-computed values of $h(i)$, i.e., for $1 \leq i < j$. It does not mean that $h(n)$ can be computed without knowing the problem size $N$ in advance.

### 1.1 Relations to Monge

In this section, we briefly introduce the definition of Monge property. See the survey [1] for more details. Consider an $N \times N$ matrix $A$. Denote by $R(n)$ the *index* of the rightmost minimum of row $n$ of $A$, i.e.,

$$R(n) = \max\{j : A_{n,j} = \min_{1 \leq i \leq N} A_{n,i}\}.$$

A matrix $A$ is *monotone* if $R(1) \leq R(2) \leq \cdots \leq R(N)$, $A$ is *totally monotone* if all submatrices[1] of $A$ are monotone. The SMAWK algorithm [2] says that if $A$ is totally monotone, then it can compute all of the $R(n)$ for $1 \leq n \leq N$ in $O(N)$ time.

For our problem, if we set

$$A_{n,j} = \begin{cases} a(n, j) & 1 \leq j \leq n \leq N \\ \infty & \text{otherwise} \end{cases} \tag{3}$$

then $h(n) = a(n, R(n))$. Hence, if we can show the matrix $A$ defined by (3) is totally monotone, then the SMAWK algorithm can solve our problem (offline version) in $O(N)$ time. Totally monotone properties are usually established by showing a slightly stronger property, the Monge Property (also known as the *quadrangle inequality*). A matrix $A$ is Monge if $\forall\ 1 \leq n < N$ and $\forall\ 1 \leq j < N$,

$$A_{n,j} + A_{n+1,j+1} \leq A_{n+1,j} + A_{n,j+1}.$$

It is easy to show that $A$ is totally monotone if it is Monge. So, for the offline version of our problem, we only need to show that the matrix $A$ defined by (3) is Monge, i.e., $\forall\ 1 \leq j < n < N$,

$$a(n, j) + a(n + 1, j + 1) \leq a(n + 1, j) + a(n, j + 1). \tag{4}$$

[1] In this paper, submatrices can take non-consecutive rows and columns from the original matrix, and are not necessarily square matrices.

By the conditions in Theorem 1,

$$a(n+1,j)+a(n,j+1)-a(n,j)-a(n+1,j+1)=(\delta_j-\delta_{j+1})\beta_{n+1}\geq 0.$$

So, the matrix $A$ defined by (3) is Monge, and the SMAWK algorithm solves the offline problem.

Our problem is a special case of Monge. But how special a case? Referring to Section 2.2 of [1] for more details, we see that if we only consider the finite entries, then a matrix $A$ is Monge if and only if $\forall\ A_{n,j}\neq\infty$,

$$A_{n,j}=P_n+Q_j+\sum_{k=n}^{N}\sum_{i=1}^{j}F_{ki} \tag{5}$$

where $P$ and $Q$ are vectors, and $F$ is an $N\times N$ matrix, called the *distribution matrix*, whose entries are all nonnegative. For our problem, let $\delta_0=\delta_1$. Then

$$\begin{aligned}a(n,j)&=a(N,j)-\sum_{k=n+1}^{N}c_k-\delta_j\sum_{k=n+1}^{N}\beta_k\\&=a(N,j)-\sum_{k=n+1}^{N}c_k-\delta_0\sum_{k=n+1}^{N}\beta_k+(\delta_0-\delta_j)\sum_{k=n+1}^{N}\beta_k\end{aligned}$$

So, in our problem,

$$P_n=-\sum_{k=n+1}^{N}(c_k+\delta_0\beta_k),\qquad Q_j=a(N,j),\qquad F_{ki}=(\delta_{i-1}-\delta_i)\beta_{k+1},$$

where we define $\beta_{N+1}=0$. This shows that our problem is a special case of the Monge property where the distribution matrix has rank 1.

Conversely, if the distribution matrix $F$ has rank 1, then the values of $a(n,j)$ satisfy the conditions of Theorem 1. So, Theorem 1 is really showing that the row minima of any Monge matrix defined by a rank 1 distribution matrix can be found online.

## 2 The Algorithm

In this section, we show the main algorithm that achieves the $O(1)$ amortized bound in Theorem 1. We will show the algorithm at step $n$, where the values of $h(i)$ have been computed for $1\leq i<n$, and we want to compute the value of $h(n)$. By the conditions in Theorem 1 and the extra condition, all the values $a(n,j)$ and $\delta_j$ for $1\leq j\leq n\leq N$ are known.

The key concept of the algorithm is a set of straight lines defined as follows.

**Definition 2.** *$\forall\ 1\leq j\leq n\leq N$, we define*

$$L_j^n(x)=a(n,j)+\delta_j\cdot x \tag{6}$$

So, $h(n) = \min_{1 \le j \le n} L_j^n(0)$. To compute $\min_{1 \le j \le n} L_j^n(x)$ at $x = 0$ efficiently, the algorithm maintains $\min_{1 \le j \le n} L_j^n(x)$ for the entire range $x \ge 0$, i.e., at step $n$, the algorithm maintains the *lower envelope* of the set of lines $\{L_j^n(x) : 1 \le j \le n\}$ in the range $x \in [0, \infty)$.

### 2.1 The Data Structure

The only data structure used is an array, called the *active-indices array*, $Z = (z_1, \ldots, z_t)$ for some length $t$. It will be used to represent the lower envelope. It stores, from left to right, the indices of the lines that appear on the lower envelope in the range $x \in [0, \infty)$. That is, at step $n$, if we walk along the lower envelope from $x = 0$ to the right, then we will sequentially encounter the lines $L_{z_1}^n(x), L_{z_2}^n(x), \ldots, L_{z_t}^n(x)$. Since $\delta_1 > \delta_2 > \cdots > \delta_n$, and by the properties of lower envelopes, we have $z_1 < z_2 < \cdots < z_t = n$, and no line can appear more than once in the active-indices array.

Once we have the active-indices array, computing $h(n)$ becomes easy as $h(n) = a(n, z_1)$. So, the problem is how to obtain the active-indices array. Inductively, when the algorithm enters step $n$ from step $n-1$, it maintains an active-indices array for step $n-1$, which represents the lower envelope of the lines $\{L_j^{n-1}(x) : 1 \le j \le n-1\}$. So, the main part of the algorithm is to *update* the old active-indices array to the new active-indices array for $\{L_j^n(x) : 1 \le j \le n\}$.

Before introducing the algorithm, we introduce another concept, the *break-point* array, $X = (x_0, \ldots, x_t)$, where $x_0 = 0$, $x_t = \infty$ and $x_i$ $(1 \le i < t)$ is the $x$-coordinate of the intersection point of lines $L_{z_i}^n(x)$ and $L_{z_{i+1}}^n(x)$. The break-point array is *not* stored explicitly, since for any $i$, the value of $x_i$ can be computed in $O(1)$ time, given the active-indices array.

### 2.2 The Main Algorithm

In step $n$, we need to consider $n$ lines $\{L_j^n(x) : 1 \le j \le n\}$. The algorithm will first deal with the $n-1$ lines $\{L_j^n(x) : 1 \le j \le n-1\}$, and then add the last line $L_n^n(x)$. Figure 1 illustrates the update process by an example. Figure 1(a) shows what we have from step $n-1$, Figure 1(b) shows the considerations for the first $n-1$ lines, and Figure 1(c) shows the adding of the last line.

**Deal with the first $n-1$ lines.** For the first $n-1$ lines $\{L_j^n(x) : 1 \le j \le n-1\}$, the key observation is the following lemma.

**Lemma 3.** *$\forall\ 1 < n \le N$ and $\forall\ x$,*

$$L_j^n(x) = L_j^{n-1}(x + \beta_n) + c_n, \qquad \forall\ 1 \le j \le n-1.$$

*Proof.* By (2) and (6),

$$\begin{aligned} L_j^n(x) &= [a(n, j) - \delta_j \beta_n] + \delta_j (x + \beta_n) \\ &= [a(n-1, j) + c_n] + \delta_j (x + \beta_n) \\ &= L_j^{n-1}(x + \beta_n) + c_n. \end{aligned}$$

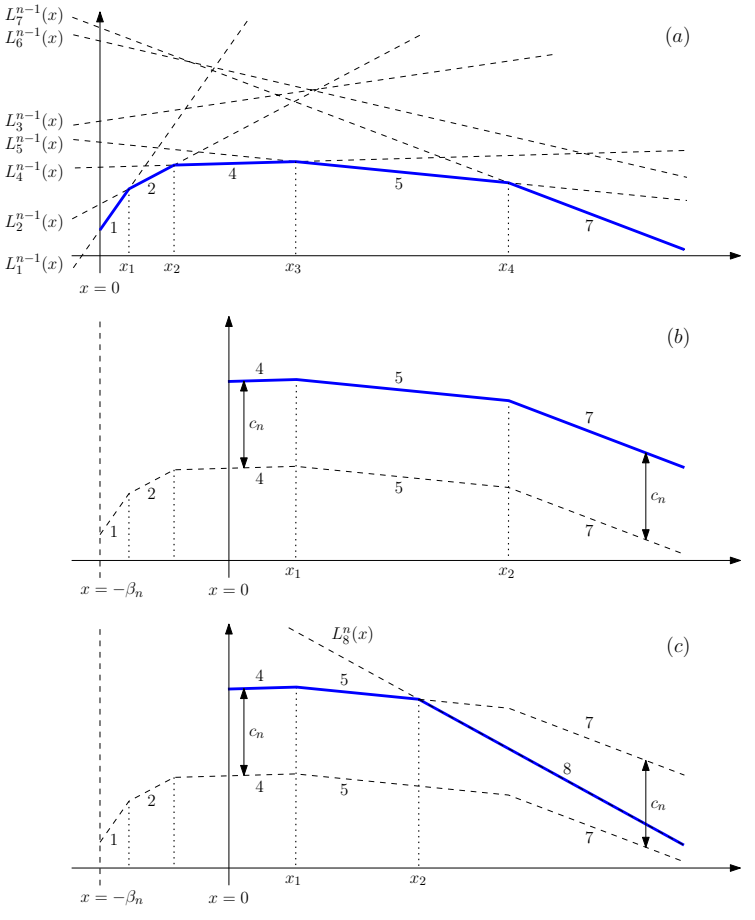



**Fig. 1.** The update of the active-indices array from Step $n-1$ to Step $n$, where $n = 8$. The thick solid chains are the lower envelopes. Figure (a) shows the lower envelope for the lines $\{L_j^{n-1}(x) : 1 \le j \le n-1\}$, Figure (b) shows the lower envelope for the lines $\{L_j^n(x) : 1 \le j \le n-1\}$, and Figure (c) shows the lower envelope for the lines $\{L_j^n(x) : 1 \le j \le n\}$. The numbers beside the line segments are the indices of the lines. The active-indices array changes from (a)$(1, 2, 4, 5, 7)$, to (b)$(4, 5, 7)$, then to (c)$(4, 5, 8)$.

Lemma 3 says that if we translate the line $L_j^{n-1}(x)$ to the left by $\beta_n$ and upward by $c_n$, then we obtain the line $L_j^n(x)$. The translation is independent of $j$, for $1 \le j \le n-1$. So,

**Corollary 4.** *The lower envelope of the lines* $\{L_j^n(x) : 1 \le j \le n-1\}$ *is the translation of the lower envelope of* $\{L_j^{n-1}(x) : 1 \le j \le n-1\}$ *to the left by* $\beta_n$ *and upward by* $c_n$.

As an example, see Figure 1, (a) and (b). From Figure 1(a) to 1(b), the entire lower envelope translates to the left by $\beta_n$ and upward by $c_n$.

We call an active-index $z_i$ *negative* if the part of $L^n_{z_i}(x)$ that appears on the lower envelope is completely contained in the range $x \in (-\infty, 0]$. By Corollary 4, to obtain the active-indices array for $\{L^n_j(x) : 1 \le j \le n-1\}$ from the old active-indices array, we only need to delete those active-indices who becomes negative due to the translation. This can be done by a simple sequential scan. We scan the old active-indices array from left to right, check each active-index whether it becomes negative. If it is, we delete it. As soon as we find the first active-index that is nonnegative, we can stop the scan, since the rest of the indices are all nonnegative.

To be precise, we scan the old active-indices array from $z_1$ to $z_t$. For each $z_i$, we compute $x_i$, the right break-point of the segment $z_i$. If $x_i < 0$, then $z_i$ is negative. Let $z_{\min}$ be the first active-index that is nonnegative, then the active-indices array for $\{L^n_j(x) : 1 \le j \le n-1\}$ is $(z_{\min}, \ldots, z_t)$.

**Adding the last line.** We now add the line $L^n_n(x)$. Recall Condition (a) in Theorem 1. Since $L^n_n(x)$ has the smallest slope over all lines, it must be the rightmost segment on the lower envelope. And since no line can appear on the lower envelope more than once, we only need to find the intersection point between $L^n_n(x)$ and the lower envelope of $\{L^n_j(x) : 1 \le j \le n-1\}$. Assume they intersect on segment $z_{\max}$, then the new lower envelope should be $(z_{\min}, \ldots, z_{\max}, n)$. See Figure 1(c), in the example, $z_{\max} = 5$.

To find $z_{\max}$, we also use a sequential scan, but from right to left. We scan the active-indices array from $z_t$ to $z_{\min}$. For each $z_i$, we compute $x_{i-1}$, the left break-point of segment $z_i$, and compare the values of $L^n_n(x_{i-1})$ and $L^n_{z_i}(x_{i-1})$. If $L^n_n(x_{i-1})$ is smaller, then $z_i$ is deleted from the active-indices array. Otherwise, we find $z_{\max}$.

**The running time.** The two sequential scans use amortized $O(1)$ time per step, since each line can be added to or deleted from the active-indices array at most once.

### 2.3 The Worst-Case Bound

To achieve the worst-case bound, we can use binary search to find $z_{\min}$ and $z_{\max}$. Since for a given index $z$ and value $x$ the function $L^n_z(x)$ can be computed in $O(1)$ time, the binary search takes $O(\log N)$ time worst case.

To keep both the $O(1)$ amortized time and the $O(\log N)$ worst-case time, we run both the sequential search and the binary search in parallel, interleaving their steps, stopping when the first one of the two searches completes.

## 3 Applications

We will now see two applications. Both will require *multiple* applications of our technique, and both will be in the form

$$H(d,n) = \min_{d-1 \le j \le n-1} \left( H(d-1,j) + W^{(d)}_{n,j} \right), \tag{7}$$

where the value of $W^{(d)}_{n,j}$ can be computed in $O(1)$ time, and the values of $H(d,n)$ for $d = 0$ or $n = d$ are given. The goal is to compute $H(D,N)$. Setting

$$a^{(d)}(n,j) = H(d-1,j) + W^{(d)}_{n,j},$$

for each fixed $d$ ($1 \le d \le D$), the values of $a^{(d)}(n,j)$ satisfy the online Monge property in Theorem 1, i.e.,

$$a^{(d)}(n,j) - a^{(d)}(n-1,j) = W^{(d)}_{n,j} - W^{(d)}_{n-1,j} = c^{(d)}_n + \delta^{(d)}_j \beta^{(d)}_n. \tag{8}$$

where $\delta^{(d)}_j$ decreases as $j$ increases, and $\beta^{(d)}_n \ge 0$.

As before, we want to compute $H(d,n)$ in online fashion, i.e., as $n$ increases from 1 to $N$, at step $n$, we want to compute the set $\mathcal{H}_n = \{H(d,n) \mid 1 \le d \le D\}$. By Theorem 1, this can be done in $O(D)$ amortized time per step. This gives a total of $O(DN)$ time to compute $H(D,N)$, while the naive algorithm requires $O(DN^2)$ time.

### 3.1 *D*-Medians on a Directed Line

The first application comes from [8]. It is the classic $D$-median problem when the underlying graph is restricted to a directed line. In this problem we have $N$ points (users) $v_1 < v_2 < \cdots < v_N$, where we also denote by $v_i$ the $x$-coordinate of the point. Each user $v_i$ has a *weight*, denoted by $w_i$, representing the amount of requests. We want to choose a subset $S \subseteq V$ as servers (medians) to provide service to the users' requests. The line is *directed*, in the sense that the requests from a user can only be serviced by a server to its left. So, $v_1$ must be a server. Denote by $\ell(v_i, S)$ the distance from $v_i$ to the nearest server to its left, i.e., $\ell(v_i, S) = \min\{v_i - v_l \mid v_l \in S, v_l \le v_i\}$. The objective is to choose $D$ servers (not counting $v_1$) to minimize the *cost*, which is $\sum_{i=1}^{N} w_i \ell(v_i, S)$.

The problem can be solved by the following DP. Let $H(d,n)$ be the minimum cost of servicing $v_1, v_2, \ldots, v_n$ using exactly $d$ servers (not counting $v_1$). Let $W_{n,j} = \sum_{l=j+1}^{n} w_l(v_l - v_{j+1})$ be the cost of servicing $v_{j+1}, \ldots, v_n$ by server $v_{j+1}$. Then

$$H(d,n) = \begin{cases} 0 & n = d \\ W_{n,0} & d = 0,\ n \ge 1 \\ \min\limits_{d-1 \le j \le n-1} (H(d-1,j) + W_{n,j}), & 1 \le d < n \end{cases}$$

The optimal cost we are looking for is $H(D,N)$.

To see the online Monge property, since

$$W_{n,j} - W_{n-1,j} = w_n(v_n - v_{j+1}),$$

we have $c_n = w_n v_n$, $\delta_j = -v_{j+1}$ and $\beta_n = w_n$, satisfying (8). So, Theorem 1 will solve the online problem in $O(D)$ amortized time per step. Hence, the total time to compute $H(D, N)$ is $O(DN)$.

[8] also gives an $O(DN)$ time algorithm, by observing the standard Monge property and applying the SMAWK algorithm. The algorithm in this paper has smaller constant factor in the $O(\cdot)$ notation, and hence is more efficient in practice. Further more, the online problem makes sense in this situation. It is known as the *one-sided* online problem. In this problem, a new user is added from right in each step. When a new user comes, our algorithm recomputes the optimal solution in $O(D)$ time amortized and $O(D \log N)$ time worst case.

We note that the corresponding online problem for solving the $D$-median on an *undirected* line was treated in [9], where a problem-specific solution was developed. The technique in this paper is a generalization of that one.

### 3.2 Wireless Mobile Paging

The second application comes from wireless networking [10]. In this problem, we are given $N$ regions, called the *cells*, and there is a *user* somewhere. We want to find which cell contains the user. To do this, we can only query a cell whether the user is in or not, and the cell will answer yes or no. For each cell $i$, we know in advance the probability that it contains the user, denote it by $p_i$. We assume $p_1 \geq p_2 \geq \cdots \geq p_N$. We also approximate the real situation by assuming the cells are *disjoint*, so $p_i$ is the probability that cell $i$ contains the user *and* no other cell does.

There is a tradeoff issue between the delay and the bandwidth requirement. For example, consider the following two strategies. The first strategy queries all cells simultaneously, while the second strategy consists of $N$ rounds, querying the cells one by one from $p_1$ to $p_N$, and stops as soon as the user is found. The fist strategy has the minimum delay, which is only one round, but has the maximum bandwidth requirement since it queries all $N$ cells. The second strategy has the maximum worst case delay of $N$ rounds, but the expected bandwidth requirement is the minimum possible, which is $\sum_{i=1}^{N} ip_i$ queries. In the tradeoff, we are given a parameter $D$, which is the worst case delay that can be tolerated, and we are going to find an optimal strategy that minimize the expected number of queries.

It is obvious that a cell with larger $p_i$ should be queried no later than one with smaller $p_i$. So, the optimal strategy actually breaks the sequence $p_1, p_2, \ldots, p_N$ into $D$ contiguous subsequences, and queries one subsequence in each round. Let $0 = r_0 < r_1 < \cdots < r_D = N$, and assume in round $i$, we query the cells from $p_{r_{i-1}+1}$ to $p_{r_i}$. Recall that the cells are disjoint. The expected number of queries, defined as the *cost*, is

$$\sum_{i=1}^{D} r_i \left( \sum_{l=r_{i-1}+1}^{r_i} p_l \right). \tag{9}$$

[10] developed a DP formulation to solve the problem. It is essentially the following DP. Let $H(d,n)$ be the optimal cost for querying cells $p_1,\ldots,p_n$ using exactly $d$ rounds. Denote $W_{n,j} = n\sum_{l=j+1}^{n} p_l$ the contribution to (9) of one round that queries $p_{j+1},\ldots,p_n$. Then

$$H(d,n) = \begin{cases} \sum_{l=1}^{n} lp_l & n = d \\ \infty & d = 0,\ n \geq 1 \\ \min\limits_{d-1\leq j\leq n-1} (H(d-1,j) + W_{n,j}), & 1 \leq d < n \end{cases}$$

[10] applied the naive approach to solve the DP in $O(DN^2)$ time. Actually, this DP satisfies the online Monge property. Since

$$W_{n,j} - W_{n-1,j} = np_n + \sum_{l=j+1}^{n-1} p_l,$$

we can set $c_n = np_n + \sum_{l=1}^{n-1} p_l$, $\delta_j = -\sum_{l=1}^{j} p_l$ and $\beta_n = 1$, satisfying (8). So, the DP can be solved in $O(DN)$ time, using either the SMAWK algorithm or the technique in this paper. However, in this problem, there is no physical interpretation to the meaning of the online situation. But, due to the simplicity of our algorithm, it runs faster than the SMAWK algorithm in practice, as suggested by the experiments in [11], and is therefore more suitable for real time applications.

## References

1. Burkard, R.E., Klinz, B., Rudolf, R.: Perspectives of Monge properties in optimization. Discrete Applied Mathematics **70**(2) (1996) 95–161
2. Aggarwal, A., Klawe, M.M., Moran, S., Shor, P.W., Wilber, R.E.: Geometric applications of a matrix-searching algorithm. Algorithmica **2** (1987) 195–208
3. Wilber, R.: The concave least-weight subsequence problem revisited. Journal of Algorithms **9**(3) (1988) 418–425
4. Eppstein, D., Galil, Z., Giancarlo, R.: Speeding up dynamic programming. In: Proceedings of the 29th Annual Symposium on Foundations of Computer Science. (1988) 488–496
5. Galil, Z., Giancarlo, R.: Speeding up dynamic programming with applications to molecular biology. Theoretical Computer Science **64**(1) (1989) 107–118
6. Galil, Z., Park, K.: A linear-time algorithm for concave one-dimensional dynamic programming. Information Processing Letters **33**(6) (1990) 309–311
7. Larmore, L.L., Schieber, B.: On-line dynamic programming with applications to the prediction of RNA secondary structure. Journal of Algorithms **12**(3) (1991) 490–515
8. Woeginger, G.J.: Monge strikes again: Optimal placement of web proxies in the Internet. Operations Research Letters **27**(3) (2000) 93–96
9. Fleischer, R., Golin, M.J., Zhang, Y.: Online maintenance of $k$-medians and $k$-covers on a line. Algorithmica **45**(4) (2006) 549–567

10. Krishnamachari, B., Gau, R.H., Wicker, S.B., Haas, Z.J.: Optimal sequential paging in cellular wireless networks. Wireless Networks **10**(2) (2004) 121–131
11. Bar-Noy, A., Feng, Y., Golin, M.J.: Efficiently paging mobile users under delay constraints. Unpublished manuscript (2006)

## A Dropping the Extra Condition

This appendix will show how to drop the condition that

– the values of $\delta_j$ can be computed in $O(1)$ time, provided that the values of $h(i)$ for $1 \leq i < j$ are known.

In real applications, this doesn't seem to be an issue. For example, in both of the applications in Section 3, the value of $\delta_j$ can easily be computed in $O(1)$ time when needed, and in neither of the applications does $\delta_j$ depend on the previously-computed values of $h(i)$ for $1 \leq i < j$. It is a theoretical issue, though, so in this appendix, we will show how to dispense with the condition.

Recall (2) from Theorem 1. It is true that we cannot compute $\delta_n$ from other values available at step $n$, since the constraints containing $\delta_n$ will only appear from step $n+1$. However, it suffices to compute $\delta_n$ at step $n+1$, since we can modify the algorithm a little bit. The only place that uses $\delta_n$ in step $n$ of the algorithm is in the addition of new line $L_n^n(x)$ to the lower envelope. After that, the algorithm computes $h(n)$ by evaluating the value of the lower envelope at $x = 0$, and then precedes to step $n+1$. So, we can postpone the addition of line $L_n^n(x)$ to the beginning of step $n+1$, after we compute $\delta_n$. To compute $h(n)$ at step $n$, we can evaluate the value of the lower envelope *without* $L_n^n(x)$ at $x = 0$, compare it with $L_n^n(0) = a(n, n)$, and take the smaller of the two. Hence, what is left is to show

**Lemma 5.** *A feasible value of $\delta_n$ can be computed in $O(1)$ time at step $n+1$.*

*Proof.* We will show an algorithm that computes $c_n$ and $\beta_n$ at step $n$, and computes $\delta_n$ at step $n+1$. There are actually many feasible solutions of $c_n$, $\beta_n$ and $\delta_j$ for (2). Consider a particular solution $c_n$, $\beta_n$ and $\delta_j$. If we set $c'_n = c_n + x\beta_n$, $\beta'_n = \beta_n$ and $\delta'_j = \delta_j - x$ for some arbitrary value $x$, then the new solution $c'_n$, $\beta'_n$ and $\delta'_j$ still satisfies (2). This gives us the degree of freedom to choose $\delta_1$. We choose $\delta_1 = 0$ and immediately get

$$c_n = a(n, 1) - a(n-1, 1), \qquad \forall\ 1 < n \leq N.$$

So, we can compute $c_n$ at step $n$.

What is left is to compute $\beta_n$ and $\delta_j$. The constraints (2) become $\forall\ 1 < j < n \leq N$,

$$\delta_j \beta_n = a(n, j) - a(n-1, j) - c_n. \tag{10}$$

$\beta_2$ does not show up in the constraints (10). In fact, the value of $\beta_2$ will not affect the algorithm. So, we can choose an arbitrary value for it, e.g. $\beta_2 = 0$. All other values, $\beta_n$ $(3 \leq n \leq N)$ and $\delta_j$ $(2 \leq j \leq N)$, appear in the constraints (10),

but we still have one degree of freedom. Consider a particular solution $\beta_n$ and $\delta_j$ to the constraints (10). If we set $\beta'_n = \beta_n/x$, and $\delta'_j = \delta_j \cdot x$ for some $x > 0$, then we obtain another feasible solution. So, we can choose $\delta_2$ to be an arbitrary negative value, e.g. $\delta_2 = -1$. The rest is easy. In step $n$, we can compute $\beta_n$ by

$$\beta_n = [a(n,2) - a(n-1,2) - c_n]/\delta_2,$$

and in step $n+1$, we compute $\delta_n$ by

$$\delta_n = [a(n+1,n) - a(n,n) - c_{n+1}]/\beta_{n+1}.$$

Hence, the lemma follows.

# Covering Many or Few Points with Unit Disks*

Mark de Berg[1], Sergio Cabello[2], and Sariel Har-Peled[3]

[1] Department of Computer Science, TU Eindhoven, the Netherlands
[2] Department of Mathematics, FMF, University of Ljubljana, and Department of Mathematics, IMFM, Slovenia
[3] Department of Computer Science, University of Illinois, USA

**Abstract.** Let $P$ be a set of $n$ weighted points. We study approximation algorithms for the following two continuous facility-location problems.

In the first problem we want to place $m$ unit disks, for a given constant $m \geqslant 1$, such that the total weight of the points from $P$ inside the union of the disks is maximized. We present a deterministic algorithm that can compute, for any $\varepsilon > 0$, a $(1-\varepsilon)$-approximation to the optimal solution in $O(n \log n + \varepsilon^{-4m} \log^{2m}(1/\varepsilon))$ time.

In the second problem we want to place a single disk with center in a given constant-complexity region $X$ such that the total weight of the points from $P$ inside the disk is minimized. Here we present an algorithm that can compute, for any $\varepsilon > 0$, with high probability a $(1+\varepsilon)$-approximation to the optimal solution in $O(n(\log^3 n + \varepsilon^{-4} \log^2 n))$ expected time.

## 1 Introduction

Let $P$ be a set of $n$ points in the plane, where each point $p \in P$ has a given weight $w_p > 0$. For any $P' \subseteq P$, let $w(P') = \sum_{p \in P'} w_p$ denote the sum of the weights over $P'$. We consider the following two geometric optimization problems:

- $\mathcal{M}(P, m)$. Here we are given a weighted point set $P$ and a parameter $m$, where $m$ is an integer constant with $m \geqslant 1$. The goal is to place $m$ unit disks that maximize the sum of the weights of the covered points. With a slight abuse of notation, we also use $\mathcal{M}(P, m)$ to denote the value of an optimal solution, that is,

$$\mathcal{M}(P, m) = \max \{ w(P \cap U) \mid U \text{ is the union of } m \text{ unit disks} \}.$$

- $\mathfrak{min}(P, X)$. Here we are given a weighted point set $P$ and a region $X$ of constant complexity in the plane. The goal is to place a single unit disk with center in $X$ that minimizes the sum of the weights of the covered points. Note that the problem is not interesting if $X = \mathbb{R}^2$. We use $\mathfrak{min}(P, X)$ as the value of an optimal solution, that is,

$$\mathfrak{min}(P, X) = \min \{ w(P \cap D) \mid D \text{ is a unit disk whose center is in } X \}.$$

* MdB was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301. SC was partially supported by the European Community Sixth Framework Programme under a Marie Curie Intra-European Fellowship, and by the Slovenian Research Agency, project J1-7218.

T. Erlebach and C. Kaklamanis (Eds.): WAOA 2006, LNCS 4368, pp. 55–68, 2006.

The problems under consideration naturally arise in the context of locational analysis, namely when considering placement of facilities that have a fixed area of influence, such as antennas or sensors. $\mathcal{M}(P, m)$ models the problem of placing $m$ of such new facilities that maximize the number of covered clients, while $\mathbf{min}(P, X)$ models the placement of a single obnoxious facility. $\mathbf{min}(P, X)$ also models the placement of a facility in an environment of obnoxious points.

*Related work and other variants.* Facility location has been studied extensively in many different variants and it goes far beyond the scope of our paper to review all the work in this area. We confine ourselves to discussing the work that is directly related to our variant of the problem. For a general overview of facility-location problems in the plane, we refer to the survey by Plastria [16].

The problem $\mathcal{M}(P, m)$ for $m = 1$ was introduced by Drezner [10]. Later Chazelle and Lee [7] gave an $O(n^2)$-time algorithm for this case. An approximation algorithm has also been given: Agarwal *et al.* [1] provided a Monte-Carlo $(1-\varepsilon)$-approximation algorithm for $\mathcal{M}(P, 1)$ when $P$ is an unweighted point set. If we replace each point $p \in P$ by a unit disk centered at $p$, then $\mathcal{M}(P, 1)$ is reduced to finding a point of maximum depth in the arrangement of disks. This implies that the recent results of Aronov and Har-Peled [2] give a Monte-Carlo $(1-\varepsilon)$-approximation algorithm that runs in $O(n\varepsilon^{-2} \log n)$ time. Both the running time and the approximation factor hold with high probability. Although the algorithm is described for the unweighted case, it can be extended to the weighted case, giving an $O(n\varepsilon^{-2} \log n \log(n/\varepsilon))$ time algorithm.

Somewhat surprisingly, the problem $\mathcal{M}(P, m)$ seems to have not been studied so far for $m > 1$. For $m = 2$, however, Cabello *et al.* [4] have shown how to solve a variant of the problem where the two disks are required to be disjoint. (This condition changes the problem significantly, because now issues related to packing problems arise.) Their algorithm runs in $O(n^{8/3} \log^2 n)$ time.

The problem $\mathbf{min}(P, X)$ was first studied by Drezner and Wesolowsky [11], who gave an $O(n^2)$-time algorithm. Note that if as before we replace each point by a unit disk, the problem $\mathbf{min}(P, X)$ is reduced to finding a point with minimum depth in an arrangement of disks restricted to $X$. This means that for unweighted points sets, we can use the results of Aronov and Har-Peled [2] to get a $(1+\varepsilon)$-approximation algorithm for $\mathbf{min}(P, X)$ in $O(n\varepsilon^{-2} \log n)$ expected time. For technical reasons, however, this algorithm cannot be trivially modified to handle weighted points.

The extension of $\mathbf{min}(P, X)$ to the problem of placing of $m$ unit disks, without extra requirements, would have a solution consisting of $m$ copies of the same disk. Hence, we restrict our attention to the case $m = 1$. (Following the paper by Cabello *et al.* [4] mentioned above one could study this problem under the condition that the disks be disjoint, but in the current paper we are interested in possibly overlapping disks.)

There are several papers studying these problems for other shapes than unit disks. The problem $\mathbf{min}(P, X)$ for unit squares—this problem was first considered by Drezner and Wesolowsky [11]—turns out to be significantly easier than for disks and one can get subquadratic exact algorithms: Katz, Kedem, and Segal

[13] gave an optimal $O(n \log n)$ algorithm that computes the exact optimum. For disks this does not seem to be possible: Aronov and Har-Peled [2] showed that for disks $\mathfrak{min}(P, X)$ and also $\mathcal{M}(P, 1)$ are 3SUM-HARD [12], that is, these problems belong to a class of problems for which no subquadratic algorithms are known. (For some problems from this class, an $\Omega(n^2)$ lower bound has been proved in a restricted model of computation.) The problem $\mathcal{M}(P, 1)$ has also been studied for other shapes [1]. We will limit our discussion to disks from now on. Our algorithms can be trivially modified to handle squares, instead of disks, or other fixed shapes of constant description; only the logarithmic factors are affected.

*Our results.* As discussed above, $\mathcal{M}(P, m)$ is 3SUM-HARD for $m = 1$ and also $\mathfrak{min}(P, X)$ is 3SUM-HARD. Since we are interested in algorithms with near-linear running time we therefore focus on approximation algorithms. For $\mathcal{M}(P, m)$ we aim to achieve $(1 - \varepsilon)$-approximation algorithms; given a parameter $\varepsilon > 0$, such algorithms compute a set of $m$ disks such that the total weight of all points in their union is at least $(1 - \varepsilon)\,\mathcal{M}(P, m)$. Similarly, for $\mathfrak{min}(P, X)$ we aim for computing a disk such that the total weight of the covered points is at most $(1 + \varepsilon)\,\mathfrak{min}(P, X)$. When stating our bounds we consider $m \geqslant 1$ to be a constant and we assume a model of computation where the floor function takes constant time.

For $\mathcal{M}(P, m)$ with $m \geqslant 1$ we give a deterministic $(1 - \varepsilon)$-approximation algorithm that runs in $O(n \log n + n\varepsilon^{-4m} \log^{2m}(1/\varepsilon))$ time. As a byproduct of our approach, we also consider an exact algorithm to compute $\mathcal{M}(P, m)$; it runs in $O(n^{2m-1} \log n)$ time. For $m = 1$, we improve [1,2]. For $m > 1$, we obtain the first near-linear time algorithms.

For $\mathfrak{min}(P, X)$ we give a randomized algorithm that runs in $O(n(\log^3 n + \varepsilon^{-4} \log^2 n))$ expected time and gives a $(1+\varepsilon)$-approximation with high probability. This is the first near-linear time approximation algorithm for this problem that can handle weighted points.

## 2 Notation and Preliminaries

It will be convenient to define a *unit disk* as a closed disk of diameter 1. Let $s := \sqrt{2}/2$, so that a square of side $s$ has diagonal of unit length and can be covered by a unit disk, and let $\Delta = 3ms$. (Recall that $m$ is the number of disks we want to place.) We assume without loss of generality that no coordinate of the points in $P$ is a multiple of $s$. For a positive integer $I$ we use the notation $[I]$ to denote the set $\{0, 1, 2, \ldots, I\}$. For a pair $(a, b) \in [3m]^2$, we use $G_{(a,b)}$ to denote the grid of spacing $\Delta$ such that $(as, bs)$ is one of the grid vertices, and we define $G := G_{(0,0)}$. We consider the cells of a grid to be open. Finally, we let $L_{(a,b)}$ denote the set of grid lines that define $G_{(a,b)}$. Thus $L_{(a,b)}$ is given by

$$\{(x, y) \in \mathbb{R}^2 \mid y = bs + k \cdot \Delta \text{ and } k \in \mathbb{Z}\} \cup \{(x, y) \in \mathbb{R}^2 \mid x = as + k \cdot \Delta \text{ and } k \in \mathbb{Z}\}$$

The following lemma follows from an easy counting argument.

**Lemma 1.** *Let $U := D_1 \cup \cdots \cup D_m$ be the union of $m$ unit disks. There is some $(a, b) \in [3m]^2$ such that $L_{(a,b)}$ does not intersect $U$ so that each disk $D_i$ is fully contained in a cell of $G_{(a,b)}$.* □

Throughout the paper we use the expression *with high probability*, or *whp* for short, to indicate that, for any given constant $c > 0$, the failure probability can be bounded by $n^{-c}$. (In our algorithms, the value $c$ affects the constant factor in the $O$-notation expressing the running time.)

An *integer-weighted* point set $Q$ is a weighted point set with integer weights. We can see $Q$ as a multiset where each point is repeated as many times as its weight. We use $P$ for arbitrary weighted point sets and $Q$ for integer-weighted point sets. A *$p$-sample* $R$ of $Q$, for some $0 \leqslant p \leqslant 1$ is obtained by adding each point of the multiset $Q$ to $R$ with probability $p$, independently. If $R$ is a $p$-sample of $Q$ and $p \cdot w(Q) \geqslant c \log n$, for an appropriate constant $c$, then it follows from Chernoff bounds that $R$ has $\Theta(p \cdot w(Q))$ points whp.

## 3 Approximation Algorithms for $\mathcal{M}(P, m)$

Our algorithm uses $(1/r)$-approximations [5,6]. In our application they can be defined as follows. Let $\mathcal{U}$ be the collection of sets $U \subset \mathbb{R}^2$ that are the union of $m$ unit disks, and let $P$ be a weighted point set. A weighted point set $A$ is a *$(1/r)$-approximation* for $P$ if for each $U \in \mathcal{U}$ we have: $|w(U \cap A) - w(U \cap P)| \leqslant w(P)/r$. The following result is due to Matoušek [15].

**Lemma 2.** *Let $P$ be a weighted point set with $n$ points and $1 \leqslant r \leqslant n$. We can construct in $O(n(r^2 \log r)^{2m})$ time a $(1/r)$-approximation $A$ for $P$ consisting of $O(r^2 \log r)$ points.* □

At first sight it may seem that this solves our problem: compute a $(1/r)$-approximation for $r = 1/\varepsilon$, and solve the problem for the resulting set of $O(\varepsilon^{-2} \log(1/\varepsilon))$ points. Unfortunately, this is not true: the error in the approximation is $w(P)/r$, not $w(U \cap P)/r$. Hence, when $w(P)$ is significantly larger than $w(U \cap P)$ we do not get a good approximation. Indeed, to obtain a good approximation we need to choose $r = w(P)/(\varepsilon \cdot \mathcal{M}(P, m))$. But now $r$ may become quite large—in fact $\Theta(n)$ in the worst case—and it seems we do not gain anything. Nevertheless, this is the route we take. The crucial fact is that, even though the size of the approximation may be $\Theta(n)$, we can still gain something: we can ensure that any cell of $G = G(0, 0)$ contains only a few points. This will allow us to compute the optimal solution within a cell quickly. By combining this with a dynamic-programming approach and using several shifted grids, we can then obtain our result. We start with a lemma guaranteeing the existence of an approximation with a few points per grid cell.

**Lemma 3.** *Let $0 < \varepsilon < 1$ be a parameter and let $P$ be a set with $n$ weighted points. Let $r := w(P)/(\varepsilon \cdot \mathcal{M}(P, m))$; note that the value of $r$ is not known. We can find in $O(n \log n + n\varepsilon^{-4m} \log^{2m}(1/\varepsilon))$ time a $(1/2r)$-approximation $A$ for $P$ consisting of at most $n$ points and such that each cell of $G$ contains $O(\varepsilon^{-2} \log(1/\varepsilon))$ points from $A$.*

*Proof.* Let $\mathcal{C}$ be the collection of cells from $G$ that contain some point of $P$. For a cell $C \in \mathcal{C}$, define $P_C := P \cap C$. Set $r' := 72m^2/\varepsilon$. For each cell $C \in \mathcal{C}$, compute a $(1/r')$-approximation $A_C$ for $P_C$. We next show that the set $A := \bigcup_{C \in \mathcal{C}} A_C$ is a $(1/2r)$-approximation for $P$ with the desired properties.

For any cell $C$ we have $w(P_C) \leqslant 9m \cdot \mathcal{M}(P, m)$ because $C$ can be decomposed into $9m$ rectangles of size $s \times ms$, and for each of these rectangles $R$ we have $w(R \cap P) \leqslant \mathcal{M}(P, m)$. Since $A_C$ is a $(1/r')$-approximation for $P_C$, we therefore have for any $U \in \mathcal{U}$,

$$|w(U \cap A_C) - w(U \cap P_C)| \leqslant \frac{w(P_C)}{r'} \leqslant \frac{9m \cdot \mathcal{M}(P, m)}{72m^2/\varepsilon} = \frac{\varepsilon}{8m} \cdot \mathcal{M}(P, m).$$

A unit disk of $U \in \mathcal{U}$ can intersect at most 4 cells of $G$, and therefore any $U \in \mathcal{U}$ can intersect at most $4m$ cells of $G$. If $\mathcal{C}_U$ denotes the cells of $G$ intersected by $U$, we have $|\mathcal{C}_U| \leqslant 4m$, so

$$\begin{aligned} |w(U \cap A) - w(U \cap P)| &= \left| \sum_{C \in \mathcal{C}_U} (w(U \cap A_C) - w(U \cap P_C)) \right| \\ &\leqslant \sum_{C \in \mathcal{C}_U} |w(U \cap A_C) - w(U \cap P_C)| \\ &\leqslant \sum_{C \in \mathcal{C}_U} \frac{\varepsilon}{8m} \cdot \mathcal{M}(P, m) \quad \leqslant (\varepsilon/2) \cdot \mathcal{M}(P, m). \end{aligned}$$

We conclude that $A$ is indeed a $(1/2r)$-approximation for $P$. For constructing the set $A$, we can classify the points $P$ by cells of $G$ in $O(n \log n)$ time, and then for each non-empty cell $C$ apply Lemma 2 to get a $(1/r')$-approximation $A_C$ for $P_C$. Since $m$ is a fixed constant, we have $r' = O(1/\varepsilon)$, and according to Lemma 2, $A_C$ will contain $O((r')^2 \log(r')) = O(\varepsilon^{-2} \log(1/\varepsilon))$ points. Also, computing $A_C$ takes $O(|P_C| \cdot (r'^2 \log r')^{2m}) = O(|P_C| \cdot (\varepsilon^{-2} \log(1/\varepsilon))^{2m})$ time, and adding the time over all cells $C \in \mathcal{C}$, we obtain the claimed running time. □

It is not hard to show that choosing the value of $r$ as in Lemma 3 indeed leads to a $(1 - \varepsilon)$-approximation.

**Lemma 4.** *Let $0 < \varepsilon < 1$ be a parameter and let $P$ be a set with $n$ weighted points. Let $A$ be a $(1/2r)$-approximation for $P$, where $r = w(P)/(\varepsilon \cdot \mathcal{M}(P, m))$. If $U_A^*$ is an optimal solution for $\mathcal{M}(A, m)$, then $w(P \cap U_A^*) \geqslant (1 - \varepsilon) \cdot \mathcal{M}(P, m)$.* □

It remains to find an optimal solution $U_A^*$ for $A$. For a point set $B$, an integer $m$, and a cell $C$, define $\mathcal{M}(B, m, C)$ to be the maximum sum of the weights of $B$ that $m$ disks inside the cell $C$ can cover. Let us assume that we have an algorithm *Exact*$(B, m, C)$—later we will provide such an algorithm—that finds the exact value $\mathcal{M}(B, m, C)$ in $T(k, m)$ time. For technical reasons, we also assume that $T(k, m)$ has the following two properties: $T(k, j) \leqslant T(k, m)$ for $j \leqslant m$ and $T(k, m)$ is superlinear but polynomially bounded for any fixed $m$. The next lemma shows that we can then compute the optimal solution for $A$ quickly, using a dynamic-programming approach.

**Lemma 5.** *Let $A$ be a point set with at most $n$ points such that each cell of $G$ contains at most $k$ points. We can find $\mathcal{M}(A,m)$ in $O(n\log n+(n/k)\cdot T(k,m))$ time.*

*Proof.* For each $(a,b)\in[3m]^2$, let $\mathcal{M}_{(a,b)}(A,m)$ be the optimal weight we can cover with $m$ unit disks that are disjoint from $L_{(a,b)}$. We have $\mathcal{M}(A,m)=\max_{(a,b)\in[3m]^2}\mathcal{M}_{(a,b)}(A,m)$ by Lemma 1. We will show how to compute each $\mathcal{M}_{(a,b)}(A,m)$ in $O(n\log n+(n/k)\cdot T(k,m))$ time, which proves our statement because $m^2=O(1)$. First we give the algorithm, and then discuss its time bound.

Consider a fixed $(a,b)\in[3m]^2$. Let $\mathcal{C}=\{C_1,\ldots,C_t\}$ be the cells of $G_{(a,b)}$ that contain some point from $P$; we have $|\mathcal{C}|=t\leqslant n$. For any cell $C_i\in\mathcal{C}$, define $A_i=A\cap C_i$.

For each cell $C_i\in\mathcal{C}$ and each $j\in\{1,\ldots,m\}$, compute $\mathcal{M}(A_i,j,C_i)$ by calling the procedure *Exact*$(A_i,j,C_i)$. From the values $\mathcal{M}(A_i,j,C_i)$ we can compute $\mathcal{M}_{(a,b)}(A,m)$ using dynamic programming across the cells of $\mathcal{C}$, as follows. Define $B_i=A_1\cup\cdots\cup A_i$. We want to compute $\mathcal{M}_{(a,b)}(B_i,j)$ for all $i,j$. To this end we note that an optimal solution $\mathcal{M}_{(a,b)}(B_i,j)$ will have $\ell$ disks inside $A_i$, for some $0\leqslant\ell\leqslant j$, and the remaining $j-\ell$ disks spread among the cells $C_1,\ldots,C_{i-1}$. This leads to the following recursive formula:

$$\mathcal{M}_{(a,b)}(B_i,j)=\begin{cases}\mathcal{M}(A_1,j,C_1) & \text{if } i=1\\ \max_{0\leqslant\ell\leqslant j}\{\mathcal{M}(A_i,\ell,C_i)+\mathcal{M}_{(a,b)}(B_{i-1},j-\ell)\} & \text{otherwise}\end{cases}$$

Since $\mathcal{M}_{(a,b)}(B_t,m)=\mathcal{M}_{(a,b)}(A,m)$, we end up computing the desired value $\mathcal{M}_{(a,b)}(A,m)$. This finishes the description of the algorithm.

The time used to compute $\mathcal{M}_{(a,b)}(A,m)$ can be bounded as follows. Firstly, observe that constructing $A_i$ for all $C_i\in\mathcal{C}$ takes $O(n\log n)$ time. For computing the values $\mathcal{M}(A_i,j,C_i)$ for all $i,j$ we need time

$$\sum_{C_i\in\mathcal{C}}\sum_{j=1}^{m}T(|A_i|,j)\leqslant\sum_{C_i\in\mathcal{C}}m\cdot T(|A_i|,m)=O\left(\sum_{C_i\in\mathcal{C}}T(|A_i|,m)\right),$$

where the first inequality follows because for $j\leqslant m$ we have $T(k,j)\leqslant T(k,m)$, and the second one follows since $m$ is a constant. We have $|A_i|\leqslant 4k$ for any $C_i\in\mathcal{C}$ because $C_i$ intersects at most 4 cells of $G$. Moreover, because $T(k,m)$ is superlinear in $k$ for fixed $m$, the sum is maximized when the points concentrate in as few sets $A_i$ as possible. Therefore, the needed time can be bounded by

$$O\left(\sum_{C_i\in\mathcal{C}}T(|A_i|,m)\right)\leqslant O\left(\sum_{i=1}^{\lceil n/4k\rceil}T(4k,m)\right)=O((n/k)\cdot T(k,m)),$$

where we have used that $T(4k,m)=O(T(k,m))$ because $T$ is polynomially bounded. Once we have the values $\mathcal{M}(A_i,j,C_i)$ for all $i,j$, the dynamic programming requires computing $O(tm)=O(n)$ values $\mathcal{M}_{(a,b)}(B_i,j)$, and each element requires $O(m)=O(1)$ time. Therefore, the dynamic programming takes $O(n)$ time. We conclude that finding $\mathcal{M}_{(a,b)}(A,m)$ takes $O(n\log n+(n/k)\cdot T(k,m))$ time for any $(a,b)\in[3m]^2$. □

Putting everything together, we obtain the following result.

**Lemma 6.** *For any weighted point set $P$ with $n$ points, we can find in time $O(n \log n + n\varepsilon^{-4m} \log^{2m}(1/\varepsilon) + (n/k) \cdot T(k, m))$ a set of $m$ disks that cover a weight of at least $(1 - \varepsilon)\,\mathcal{M}(P, m)$, where $k = O(\varepsilon^{-2} \log(1/\varepsilon))$.*

*Proof.* Given $P$ and a parameter $\varepsilon$, consider the (unknown) value $r = \frac{w(P)}{\varepsilon \cdot \mathcal{M}(P,m)}$. We use Lemma 3 to compute a point set $A$ with at most $n$ points and such that $A$ is a $(1/2r)$-approximation for $P$ and any cell of $G$ contains $O(\varepsilon^{-2} \log(1/\varepsilon))$ points.

We then use Lemma 5 to find an optimal solution $U_A^*$ for $\mathcal{M}(A, m)$. It takes $O(n \log n + (n/k) \cdot T(k, m))$ time, where $k = O(\varepsilon^{-2} \log(1/\varepsilon))$. From Lemma 4, we know that $w(U_A^* \cap P) \geqslant (1 - \varepsilon)\,\mathcal{M}(P, m)$, and the result follows. □

Theorem 2 below states there is an algorithm for the exact problem that uses $T(k, m) = O(k^{2m-1} \log k)$ time for $m > 1$. For $m = 1$, we have $T(k, 1) = O(k^2)$ because of Chazelle and Lee [7]. We can then use the previous lemma to obtain our final result.

**Theorem 1.** *Let $m \geqslant 1$ be a fixed positive integer constant. Given a parameter $0 < \varepsilon < 1$ and a weighted point set $P$ with $n$ points, we can find a set of $m$ disks that cover a weight of at least $(1 - \varepsilon)\,\mathcal{M}(P, m)$ in time $O(n \log n + n\varepsilon^{-4m} \log^{2m}(1/\varepsilon))$ time.* □

*Exact algorithms for $\mathcal{M}(P, m, C)$.* We want to find the set of $m$ disks contained in a cell $C$ of a grid that maximize the sum of the weights of the covered points. Let $X$ be the set of possible centers for a unit disk contained in $C$—the domain $X$ is simply a square with the same center as $C$ and of side length $\Delta - 1$ instead of $\Delta$.

For a point $p \in P$, let $D_p$ be the unit disk centered at $p$. The weight of $D_p$ is $w_p$, the weight of $p$. Let $\mathcal{D}_P := \{D_p : p \in P\}$ be the set of all disks defined by $P$. For a point $q \in \mathbb{R}^2$ and a set $\mathcal{D}$ of weighted disks, we define $\mathrm{depth}(q, \mathcal{D})$ to be the sum of the weights of the disks from $\mathcal{D}$ that contain $q$. Let $\mathcal{A}$ denote the arrangement induced by the disks from $\mathcal{D}_P$. For any point $q$ inside a fixed cell $c$ of $\mathcal{A}$, the function $\mathrm{depth}(q, \mathcal{D}_P)$ is constant; we denote its value by $\mathrm{depth}(c, \mathcal{D}_P)$. Because each disk $D_p$ has the same size, the arrangement $\mathcal{A}$ can be constructed in $O(n^2)$ time [7]. Moreover, a simple traversal of $\mathcal{A}$ allows us to compute $\mathrm{depth}_P(c)$ for all cells $c \in \mathcal{A}$ in $O(n^2)$ time.

Let $V_{\mathcal{A}}$ be the set of vertices of $\mathcal{A}$, let $V_X$ be the intersection points of the boundary of $X$ with the boundary of some disk $D_p$, $p \in P$, and let $V_{\text{left}}$ be set of leftmost points from each disk $D_p$, $p \in P$. Finally, let $V = (V_{\mathcal{A}} \cup V_X \cup V_{\text{left}}) \cap X$. See Figure 1, left. If $V = \emptyset$, then $X$ is contained in some cell of $\mathcal{A}$ and the problem can trivially be solved. Otherwise we have

$$\mathcal{M}(P, m, C) = \max\{w(P \cap U) \mid U \text{ union of } m \text{ unit disks with centers at } V\},$$

that is, we only need to consider disks whose centers are in $V$. Based on this observation, we can solve $\mathcal{M}(P, m, C)$ for $m > 1$. We first consider the case $m = 2$.

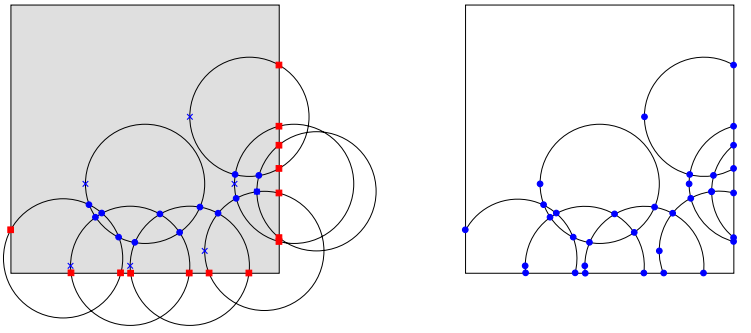


**Fig. 1.** Left: Example showing the points $V$. The dots indicate $V_{\mathcal{A}} \cap X$, the squares indicate $V_X$, and the crosses indicate $V_{\text{left}} \cap X$. Right: planar graph $G_V$ with $V$ as vertices and connected using portions of $\mathcal{A}$ or the boundary of $X$ as edges.

**Lemma 7.** *We can compute* $\mathcal{M}(P, 2, C)$ *in* $O(n^3 \log n)$ *time.*

*Proof.* Our approach is similar to the one used by Katz and Sharir [14]. Let $\mathcal{A}^*$ the arrangement induced by the set $\mathcal{D}_P$ of disks and the sets $X$ and $V$. Let $G$ be the plane graph obtained by considering the restriction of $\mathcal{A}^*$ to $X$: the vertices of $G$ are the vertices of $\mathcal{A}^*$ contained in $X$ and the edges of $G$ are the edges of $\mathcal{A}^*$ fully contained in $X$—see Figure 1, right. For simplicity, let's assume that each vertex in $G$ has degree 4, meaning that no three points of $P$ are cocircular. This condition can be lifted at the cost of making the discussion more tedious, but without extra ideas. Consider a spanning tree of $G$ and double each edge to obtain an Euler path $\pi$. The path $\pi$ has $O(n^2)$ edges and it visits each vertex of $V$ at least once and at most four times.

The idea of the algorithm is as follows. We want to find two vertices $q, v \in V$, such that $P \cap (D_q \cup D_v)$ has maximum weight. If we fix $q$ and let $\mathcal{D}_P(q) \subset \mathcal{D}_P$ denote the disks in $\mathcal{D}_P$ *not* containing $q$, then the best pair $q, v$ (for this choice of $q$) covers a weight of $\text{depth}(q, \mathcal{D}_P) + \max_{v \in V} \text{depth}(v, \mathcal{D}_P(q))$. So our approach is to walk along the tour $\pi$ to visit all possible vertices $q \in V$, and maintain the set $\mathcal{D} := \mathcal{D}_P(q)$—we call this the set of *active disks*—such that we can efficiently perform the following operations: (i) report a vertex $v \in V$ maximizing $\text{depth}(v, \mathcal{D})$, and (ii) insert or delete a disk into $\mathcal{D}$. Then we can proceed as follows. Consider two vertices $q', q''$ that are connected by an edge of $\pi$. The sets $\mathcal{D}_P(q')$ and $\mathcal{D}_P(q'')$ of active disks can differ by at most two disks. So while we traverse $\pi$, stepping from a vertex $q'$ to an adjacent one $q''$ along an edge of $\pi$, we can update $\mathcal{D}$ with at most two insertions/deletions, and then report a vertex $v \in V$ maximizing $\text{depth}(v, \mathcal{D})$. Next we show how to maintain $\mathcal{D}$ such that both operations—reporting and updating—can be performed in $O(n \log n)$ time. Since $\pi$ has $O(n^2)$ vertices, the total time will then be $O(n^3 \log n)$, as claimed.

The main problem in maintaining the set of active disks $\mathcal{D}$ is that the insertion or deletion of a disk can change $\text{depth}(v, \mathcal{D})$ for $\Theta(n^2)$ vertices $v \in V$. Hence,

to obtain $O(n \log n)$ update time, we cannot maintain all the depths explicitly. Instead we do this implicitly, as follows.

Let $\mathcal{T}$ be a balanced binary tree on the path $\pi$, where the leftmost leaf stores the first vertex of $\pi$, the next leaf the second vertex of $\pi$, and so on. Thus the tree $\mathcal{T}$ has $O(n^2)$ nodes. For an internal node $\nu$ we denote by $\mathcal{T}_\nu$ the subtree of $\mathcal{T}$ rooted at $\nu$. Furthermore, we define $\pi(\nu)$ to be the subpath of $\pi$ from the leftmost vertex in $\mathcal{T}_\nu$ to the rightmost vertex in $\mathcal{T}_\nu$. Note that if $\mu_1$ and $\mu_2$ are the children of $\nu$, then $\pi(\nu)$ is the concatenation of $\pi(\mu_1)$ and $\pi(\mu_2)$. Also note that $\pi(\text{root}(\mathcal{T})) = \pi$. Finally, note that any subpath from $\pi$ can be expressed as the concatenation of the subpaths $\pi(\nu_1), \pi(\nu_2), \ldots$ of $O(\log n)$ nodes—this is similar to the way a segment tree [9] works.

Now consider some disk $D_p \in \mathcal{D}_P$. Since $D_p$ has $O(n)$ vertices from $V$ on its boundary, the part of $\pi$ inside $D_p$ consists of $O(n)$ subpaths. Hence, there is a collection $N(D_p)$ of $O(n \log n)$ nodes in $\mathcal{T}$—we call this set the *canonical representation* of $D_p$—such that $\pi \cap D_p$ is the disjoint union of the set of paths $\{\pi(\nu) : \nu \in N(D_p)\}$. We store at each node $\nu$ the following two values:

- Cover($\nu$): the total weight of all disks $D_p \in \mathcal{D}$ (that is, all active disks) such that $\nu \in N(D_p)$.
- MaxDepth($\nu$): the value $\max\{\text{depth}(v, \mathcal{D}(\nu)) : v \in \pi(\nu)\}$, where $\mathcal{D}(\nu) \subset \mathcal{D}$ is the set of all active disks whose canonical representation contains a node $\mu$ in $\mathcal{T}_\nu$.

Notice that $\text{MaxDepth}(\text{root}(\mathcal{T})) = \max_{v \in V} \text{depth}(v, \mathcal{D})$, so MaxDepth(root($\mathcal{T}$)) is exactly the value we want to report. Hence, it remains to describe how to maintain the values Cover($\nu$) and MaxDepth($\nu$) when $\mathcal{D}$ is updated. Consider the insertion of a disk $D_p$ into $\mathcal{D}$; deletions are handled similarly. First we find in $O(n \log n)$ time the set $N(D_p)$ of nodes in $\mathcal{T}$ that forms the canonical representation of $D_p$. The values Cover($\nu$) and MaxDepth($\nu$) are only influenced for nodes $\nu$ that are in $N(D_p)$, or that are an ancestor of such a node. More precisely, for $\nu \in N(D_p)$, we need to add the weight of $D_p$ to Cover($\nu$) and to MaxDepth($\nu$). To update the values at the ancestors we use that, if $\mu_1$ and $\mu_2$ are the children of a node $\nu$, then we have

$$\text{MaxDepth}(\nu) = \text{Cover}(\nu) + \max(\text{MaxDepth}(\mu_1), \text{MaxDepth}(\mu_2)).$$

This means we can update the values in a bottom-up fashion in $O(1)$ time per ancestor, so in $O(n \log n)$ time in total. This finishes the description of the data structure—see Katz et al [13] or Bose et al [3] for similar ideas, or how to reduce the space requirements. □

**Theorem 2.** *For fixed $m > 1$, we can compute $\mathcal{M}(P, m, C)$ in $O(n^{2m-1} \log n)$ time.*

*Proof.* For $m > 2$, fix any $m - 2$ vertices $v_1, \ldots, v_{m-2} \in V$, compute the point set $P' = P \setminus (D_{v_1} \cup \cdots \cup D_{v_{m-2}})$, and compute $\mathcal{M}(P', 2, C)$ in $O(n^3 \log n)$ time using the previous lemma. We obtain a placement of disks covering a weight of $w(P \setminus P') + \mathcal{M}(P', 2, C)$. This solution is optimal under the assumption that the

first $m-2$ disks are placed at $v_1, \ldots, v_{m-2}$. Iterating this procedure over the $O(|V|^{m-2}) = O(n^{2m-4})$ possible tuples of vertices $v_1, \ldots, v_{m-2}$, it is clear that we obtain the optimal solution for placing $m$ disks inside $C$. The time we spend can be bounded as $O(n^{2m-4}n^3 \log n) = O(n^{2m-1} \log n)$. □

## 4 Approximation Algorithms for $\min(P, X)$

We now turn our attention to the problem $\min(P, X)$ where we wish to place a single disk $D$ in $X$ so as to minimize the sum of the weights of the points in $P \cap D$. The approach consists of two stages. First, we make a binary search to find a value $T$ that is a constant factor approximation for $\min(P, X)$. For this to work, we give a decision procedure that drives the search for the value $T$. Second, we compute a $(1+\varepsilon)$-approximation of $\min(P, X)$ using a random sample of appropriate density. In both cases, we will use the following lemma; a similar result was obtained by Aronov and Har-Peled [2]. Recall that $D_a$ denotes the unit disk centered at a point $a$.

**Lemma 8.** *Let $Q$ be an* unweighted *point set with at most $n$ points, let $X$ be a domain of constant complexity, let $A$ be a set of at most $n$ points, and let $\kappa$ be a non-negative integer. We can decide in $O(n\kappa + n \log n)$ expected time if $\min\left(Q, X \setminus \left(\bigcup_{a \in A} D_a\right)\right) \leqslant \kappa$ or $\min\left(Q, X \setminus \left(\bigcup_{a \in A} D_a\right)\right) > \kappa$. In the former case we can also find a unit disk $D$ that is optimal for $\min\left(Q, X \setminus \left(\bigcup_{a \in A} D_a\right)\right)$. The running time is randomized, but the result is always correct.*

*Proof.* Let $\mathcal{A}$ be the arrangement induced by the $O(n)$ disks $D_a$, $a \in A$, and $D_q$, $q \in Q$, and let $\mathcal{A}_\kappa$ be the portion of $\mathcal{A}$ that has depth at most $\kappa$. The portion $\mathcal{A}_\kappa$ has complexity $O(n\kappa)$ [17] and it can be constructed using a randomized incremental construction, in $O(n\kappa + n \log n)$ expected time [8]. Then, we just discard all the cells of $\mathcal{A}_\kappa$ that are covered by any disk $D_a$ with $a \in A$, and for the remaining cells we check if any has depth over $\kappa$ and intersects $X$. Since $X$ has constant complexity, in each cell we spend time proportional to its complexity, and the result follows. □

The following combinatorial lemma is very similar to [2, Lemma 3.1].

**Lemma 9.** *Let $Q$ be an integer-weighted point set with $n$ points, let $X$ be any domain, and let $\Delta_Q = \min(Q, X)$. Given a value $k$, set $p = \min\{1, ck^{-1} \log n\}$, where $c > 0$ is an appropriate constant. If $R$ is a $p$-sample of $Q$ and $\Delta_R = \min(R, X)$, then whp it holds*

**(i)** *if $\Delta_Q \geqslant k/2$, then $\Delta_R \geqslant kp/4$;*
**(ii)** *if $\Delta_Q \leqslant 2k$, then $\Delta_R \leqslant 3kp$;*
**(iii)** *if $\Delta_Q \notin [k/8, 6k]$, then $\Delta_R \notin [kp/4, 3kp]$.*

□

The idea for the decision version is to distinguish between heavy and light points. The heavy points have to be avoided, while the light ones can be approximated by a set of $n$ *integer-weighted* points. Then we can use the previous lemma to decide.

**Lemma 10.** *Let $X$ be a domain with constant complexity. Given a weighted point set $P$ with $n$ points and a value $T$, we can decide in $O(n \log^2 n)$ expected time whether (i) $\mathbf{min}(P, X) < T$, or (ii) $\mathbf{min}(P, X) > 2T$, or (iii) $\mathbf{min}(P, X) \in (T/10, 10T)$, where the decision is correct whp.*

*Proof.* First we describe the algorithm then discuss its running time and finally show its correctness.
*Algorithm.* We compute the sets $A = \{p \in P \mid w_p > 2T\}$ and $\tilde{P} = P \setminus A$, as well as the domain $Y = X \setminus \bigcup_{a \in A} D_a$. If $Y = \emptyset$, then we can report $\mathbf{min}(P, X) > 2T$, since any disk with center in $X$ covers some point with weight at least $2T$. If $Y \neq \emptyset$, we construct the integer-weighted point set $Q$ obtained by picking each point from $\tilde{P}$ with weight $\lfloor 2nw_p/T \rfloor$. Define $k = 2n$, and $p = \min\{1, ck^{-1} \log n\}$, where $c$ is the same constant as in the previous lemma. We compute a $p$-sample $R$ of $Q$, and decide as follows: If $\mathbf{min}(R, Y) < kp/4$ we decide $\mathbf{min}(P, X) < T$; if $\mathbf{min}(R, Y) > 3kp$ we decide $\mathbf{min}(P, X) > 2T$; otherwise we decide $\mathbf{min}(P, X) \in (T/10, 10T)$.

*Running time.* We can compute $A, \tilde{P}, Q, R$ in linear time, and check if $Y = \emptyset$ in $O(n \log n)$ time by constructing $\bigcup_{a \in A} D_a$ explicitly using a randomized incremental construction. Each point in $\tilde{P}$ has weight at most $2T$, and therefore a point in $Q$ has an integer weight bounded by $\lfloor 2n \cdot 2T/T \rfloor = O(n)$. We conclude that $Q$ is an integer-weighted point set with $n$ points and weight $w(Q) = O(n^2)$. Since $p = O(n^{-1} \log n)$ and $kp = O(\log n)$, it follows that $R$ has $\Theta(w(Q) \cdot p) = \Theta(n \log n)$ points whp.

Because $Y = X \setminus \bigcup_{a \in A} D_a$, we can use Lemma 8 to find if $\Delta_R = \mathbf{min}(R, Y) > 3kp$ or otherwise compute $\Delta_R$ exactly, in $O(|R| \log |R| + |R|kp) = O(n \log^2 n)$ time. The running time follows.

*Correctness.* We next show that, whp, the algorithm gives a correct answer. For any unit disk $D$ centered in $Y$ we have

$$w(D \cap P) - T/2 \leqslant \sum_{p \in D \cap P} \left( w_p - \frac{T}{2n} \right) \leqslant \sum_{p \in D \cap P} \left\lfloor \frac{2nw_p}{T} \right\rfloor \cdot \frac{T}{2n} = w(D \cap Q) \cdot \frac{T}{2n} \quad (1)$$

and

$$w(D \cap Q) \cdot \frac{T}{2n} = \sum_{p \in D \cap P} \left\lfloor \frac{2nw_p}{T} \right\rfloor \cdot \frac{T}{2n} \leqslant \sum_{p \in D \cap P} \frac{2nw_p}{T} \cdot \frac{T}{2n} = w(D \cap P). \quad (2)$$

We conclude that $w(D \cap P) - T/2 \leqslant (T/2n) \cdot w(D \cap Q) \leqslant w(D \cap P)$. Using the notation $\Delta_R = \mathbf{min}(R, Y)$, $\Delta_Q = \mathbf{min}(Q, Y)$, and $\Delta_P = \mathbf{min}(P, Y)$, we have

$$\Delta_P - T/2 \leqslant \frac{T}{2n} \cdot \Delta_Q \leqslant \Delta_P. \quad (3)$$

The value $\Delta_R$ provides us information as follows:

- If $\Delta_R < kp/4$, then $\Delta_Q < k/2 = n$ whp because of Lemma 9(i). We then have
$$\Delta_P \leqslant \frac{T}{2n} \Delta_Q + \frac{T}{2} < \frac{T}{2n} \cdot n + \frac{T}{2} = T.$$

– If $\Delta_R > 3kp$, then $\Delta_Q > 2k = 4n$ whp because of Lemma 9(ii). We then have
$$\Delta_P \geqslant \frac{T}{2n} \cdot \Delta_Q > \frac{T}{2n} 4n = 2T.$$
– If $\Delta_R \in [kp/4, 3kp]$, then $\Delta_Q \in [k/8, 6k] = [n/4, 12n]$ whp by Lemma 9(iii). We then have
$$\Delta_P \leqslant \frac{T}{2n} \cdot \Delta_Q + T/2 < 10T \quad \text{and} \quad \Delta_P \geqslant \frac{T}{2n} \cdot \Delta_Q \geqslant \frac{T}{2n} > \frac{T}{10}.$$

It follows that the algorithm gives the correct answer whp. □

**Lemma 11.** *Let $X$ be a domain with constant complexity. Given a weighted point set $P$ with $n$ points, we can find in $O(n \log^3 n)$ expected time a value $T$ that, whp, satisfies $T/10 < \mathbf{min}(P, X) < 10T$.*

*Proof.* The idea is to make a binary search. For this, we will use the previous lemma for certain values $T$. Note that, if at any stage, the previous lemma returns that $\mathbf{min}(P, X) \in (T/10, 10T)$, then we have found our desired value $T$, and we can finish the search. In total, we will make $O(\log n)$ calls to the procedure of Lemma 10, and therefore we obtain the claimed expected running time. Also, the result is correct whp because we make $O(\log n)$ calls to procedures that are correct whp.

Define the interval $I_p = [w_p, (n+1) \cdot w_p)$ for any point $p \in P$, and let $I = \bigcup_{p \in P} I_p$. It is clear that $\mathbf{min}(P, X) \in I$, since the weight of the heaviest point covered by an optimal solution can appear at most $n$ times in the solution. Consider the values $B = \{w_p, (n+1) \cdot w_p \mid p \in P\}$, and assume that $B = \{b_1, \ldots, b_{2n}\}$ is sorted increasingly. Note that for the (unique) index $i$ such that $\mathbf{min}(P, X) \in [b_i, b_{i+1})$, it must hold that $b_{i+1} \leqslant (n+1) b_i$

We first perform a binary search to find two consecutive elements $b_i, b_{i+1}$ such that $\mathbf{min}(P, X) \in [b_i, b_{i+1})$. Start with $\ell = 1$ and $r = 2n$. While $r \neq \ell + 1$, set $m = \lfloor (\ell + r)/2 \rfloor$ and use the previous lemma with $T = b_m$:

– if $\mathbf{min}(P, X) < T$, then set $r = m$.
– if $\mathbf{min}(P, X) > 2T$, then set $\ell = m$.
– if $T/10 < \mathbf{min}(P, X) < 10T$, then we just return $T$ as the desired value.

Note that during the search we maintain the invariant $\mathbf{min}(P, X) \in [b_\ell, b_r)$. Since we end up with two consecutive indices $\ell = i, r = i + 1$, it must hold that $\mathbf{min}(P, X) \in [b_i, b_{i+1})$.

Next, we perform another binary search in the interval $[b_i, b_{i+1})$ as follows. Start with $\ell = b_i$ and $r = b_{i+1}$. While $r/\ell > 10$, set $m = (\ell + r)/2$ and call the procedure of Lemma 10 with $T = m$:

– if $\mathbf{min}(P, X) < T$, then set $r = m$.
– if $\mathbf{min}(P, X) > 2T$, then set $\ell = m$.
– if $T/10 < \mathbf{min}(P, X) < 10T$, then we just return $T$ as the desired value.

Since $b_{i+1} \leqslant (n+1)b_i$, it takes $O(\log n)$ iterations to ensure that $r/\ell \leqslant 10$. During the search we maintain the invariant that $\mathbf{min}(P, X) \in [\ell, r)$, and therefore we can return the last value $\ell$ as satisfying $\mathbf{min}(P, X) \in (\ell/10, 10\ell)$. □

When we have a constant factor approximation to the value $\mathbf{min}(P, X)$, we can then round the weights accordingly and take a random sample of appropriate size to obtain a $(1+\varepsilon)$-approximation. The precise statement is as follows.

**Lemma 12.** *Let $0 < \varepsilon < 1$ be a parameter, let $P$ be a weighted point set with $n$ points, and let $T$ be a value such that $T/10 < \mathbf{min}(P, X) < 10T$. We can find in $O((n/\varepsilon^4)\log^2 n)$ expected time a unit disk $D$ that, whp, satisfies $w(D \cap P) \leqslant (1+\varepsilon)\,\mathbf{min}(P, X)$.*

*Proof.* First we describe the algorithm, then show its correctness, and finally discuss its running time. The ideas are similar to the ones used in Lemma 10. However, now we also need to take into account the parameter $\varepsilon$.

*Algorithm.* We compute the sets $A = \{p \in P \mid w_p > 10T\}$ and $\tilde{P} = P \setminus A$, as well as the domain $Y = X \setminus \bigcup_{a\in A} D_a$. Since $\mathbf{min}(P, X) < 10T$ by hypothesis, we know that $\mathbf{min}(P, X) = \mathbf{min}(\tilde{P}, Y) = \mathbf{min}(P, Y)$, because any disk with center in $\bigcup_{a\in A} D_a$ covers some point of $A$. We construct an integer-weighted point set $Q$ by picking each point from $\tilde{P}$ with weight $\lfloor 20nw_p/\varepsilon T\rfloor$. Define $k = \lfloor 20n/\varepsilon\rfloor$, and let $p = \min\{1, ck^{-1}\varepsilon^{-2}\log n\}$, where $c$ is an appropriate constant. Finally, compute a $p$-sample $R$ of $Q$, find a best disk $D_R$ for $\mathbf{min}(R, X)$, and report the disk $D_R$ as solution.

*Correctness.* The correctness is seen using the same approach as in Lemma 9 and [2, Lemma 3.1].

*Running time.* For the running time, observe that $A, \tilde{P}, Q, R, Y$ can be computed in $O(n \log n)$ time, like in Lemma 10. Each point in $\tilde{P}$ has weight at most $10T$, and therefore each point of $Q$ has an integer weight bounded by $\lfloor 20n \cdot 10T/\varepsilon T\rfloor = O(n/\varepsilon)$. We conclude that $Q$ is an integer-weighted point set with $n$ points and total weight $w(Q) = O(n^2/\varepsilon)$. Note that $p = O(n^{-1}\varepsilon^{-1}\log n)$ and $kp = O(\varepsilon^{-2}\log n)$. Therefore, the set $R$ has $O(w(Q)\cdot p) = O((n/\varepsilon^2)\log n)$ points whp.

Using Chernoff bound, one can see that whp $\Delta_R = O(p\,\Delta_Q)$. Substituting $p$ and using that $\Delta_Q \leqslant \frac{20n}{\varepsilon T}\Delta_P$, we obtain

$$\Delta_R = O\left(n^{-1}\varepsilon^{-1}\log n\,\frac{20n}{\varepsilon T}\Delta_P\right) = O\left(\frac{\Delta_P \log n}{\varepsilon^2 T}\right) = O\left(\varepsilon^{-2}\log n\right).$$

Since $Y = X \setminus \bigcup_{a\in A} D_a$, we can use Lemma 8 to find a best disk for $\mathbf{min}(R, Y)$ in

$$O(|R|\log|R| + |R|\Delta_R) = O(n\varepsilon^{-2}\log n\log(n/\varepsilon) + n\varepsilon^{-4}\log^2 n) = O(n\varepsilon^{-4}\log^2 n)$$

expected time, as the lemma claims. □

By applying Lemma 11 to obtain a constant factor approximation, and then Lemma 12 to obtain a $(1+\varepsilon)$-approximation, we obtain our final result.

**Theorem 3.** *Given a domain $X$ of constant complexity, a parameter $0 < \varepsilon < 1$, and a weighted point set $P$ with $n$ points, we can find in $O(n(\log^3 n + \varepsilon^{-4} \log^2 n))$ expected time a unit disk that, with high probability, covers a weight of at most $(1 + \varepsilon)\,\mathsf{min}(P, X)$.* □

## References


1. P. K. Agarwal, T. Hagerup, R. Ray, M. Sharir, M. Smid, and E. Welzl. Translating a planar object to maximize point containment. In *ESA 2002*, LNCS 2461, 2002.
2. B. Aronov and S. Har-Peled. On approximating the depth and related problems. In *SODA 2005*, pages 886–894, 2005.
3. P. Bose, M. van Kreveld, A. Maheshwari, P. Morin, and J. Morrison. Translating a regular grid over a point set. *Comput. Geom. Theory Appl.*, 25:21–34, 2003.
4. S. Cabello, J. M. Díaz Báñez, C. Seara, J.A. Sellarès, J. Urrutia, and I. Ventura. Covering point sets with two disjoint disks or squares. Manuscript available at `http://www.fmf.uni-lj.si/~cabello/publications/`. Preliminary version appeared at EWCG'05.
5. B. Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press, New York, 2001.
6. B. Chazelle. The discrepancy method in computational geometry. In *Handbook of Discrete and Computational Geometry*, pages 983–996. CRC Press, 2004.
7. B. Chazelle and D. T. Lee. On a circle placement problem. *Computing*, 36:1–16, 1986.
8. K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.
9. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.
10. Z. Drezner. On a modified one-center model. *Management Science*, 27:848–851, 1991.
11. Z. Drezner and G. O. Wesolowsky. Finding the circle or rectangle containing the minimum weight of points. *Location Science*, 2:83–90, 1994.
12. A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, 5:165–185, 1995.
13. M. J. Katz, K. Kedem, and M. Segal. Improved algorithms for placing undesirable facilities. *Computers and Operations Research*, 29:1859–1872, 2002.
14. M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM J. Computing*, 26:1384–1408, 1997.
15. J. Matoušek. Approximations and optimal geometric divide-an-conquer. *J. Comput. Syst. Sci.*, 50:203–208, 1995.
16. F. Plastria. Continuous covering location problems. In H. Hamacher and Z. Drezner, editors, *Location Analysis: Theory and Applications*, chapter 2, pages 39–83. Springer, 2001.
17. M. Sharir. On $k$-sets in arrangements of curves and surfaces. *Discrete Comput. Geom.*, 6:593–613, 1991.

# On the Minimum Corridor Connection Problem and Other Generalized Geometric Problems*

Hans Bodlaender[1], Corinne Feremans[2], Alexander Grigoriev[2], Eelko Penninkx[1], René Sitters[3], and Thomas Wolle[4]

[1] Institute of Information and Computing Sciences, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, The Netherlands
{hansb,penninkx}@cs.uu.nl
[2] Department of Quantitative Economics, Maastricht University, P.O.Box 616, 6200 MD Maastricht, The Netherlands
{c.feremans,a.grigoriev}@ke.unimaas.nl
[3] Department of Algorithms and Complexity, Max-Planck-Institute for Computer Science, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
sitters@mpi-inf.mpg.de
[4] National ICT Australia Ltd**, Locked Bag 9013, Alexandria NSW 1435, Australia
thomas.wolle@nicta.com.au

**Abstract.** In this paper we discuss the complexity and approximability of the minimum corridor connection problem where, given a rectilinear decomposition of a rectilinear polygon into "rooms", one has to find the minimum length tree along the edges of the decomposition such that every room is incident to a vertex of the tree. We show that the problem is strongly NP-hard and give an subexponential time exact algorithm. For the special case of $k$-outerplanar graphs the running time becomes $O(n^3)$. We develop a polynomial time approximation scheme for the case when all rooms are fat and have nearly the same size. When rooms are fat but are of varying size we give a polynomial time constant factor approximation algorithm.

**Keywords:** minimum corridor connection, generalized geometric problems, complexity, exact algorithms, approximations.

## 1 Introduction

**MCC and other generalized geometric problems.** We consider the following geometric problem. Given a rectilinear decomposition of a rectilinear polygon (a subdivision into $n$ "*rooms*"), find the minimum length tree ("*corridor*") along

* This work was supported by the Netherlands Organisation for Scientific Research NWO (project *Treewidth and Combinatorial Optimisation*).
** National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

T. Erlebach and C. Kaklamanis (Eds.): WAOA 2006, LNCS 4368, pp. 69–82, 2006.

the edges of the decomposition ("*walls*") such that every room is incident to a vertex of the tree (has access to the corridor); for an illustration see Figure 1 which is borrowed from [6]. Let us refer to this problem as the *minimum corridor connection* (MCC) problem.

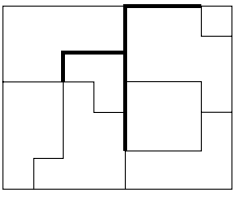


**Fig. 1.** Minimum length tree along the corridors to connect the rooms

This problem belongs to the class of so-called *generalized geometric problems* where given a collection of objects in the plane, one has to find a minimum length network satisfying certain properties that hits each object at least once. In particular, MCC can be viewed as a special case of the generalized geometric Steiner tree problem where given a set of disjoint groups of points in the plane, the problem is to find a shortest (in some metric space) interconnection tree which includes at least one point from each group.

The most studied generalized geometric problem is the following generalization of the classic Euclidean Traveling Salesman Problem (ETSP). Assume that a salesman has to visit $n$ customers. Each customer has a set of specified locations in the plane (referred to as a *region* or *neighborhood*) where the customer is willing to meet the salesman. The objective is to find a shortest (closed) salesman tour visiting each customer. If each region is a single point, the problem becomes the classic ETSP. This described generalization of ETSP is known as the Generalized ETSP [12], or Group-ETSP, or ETSP with neighborhoods [5,8], or the Geometric Covering Salesman Problem [1]. For short, we shall refer to this problem as GTSP. In a similar way one can define generalizations for Minimum Steiner Tree (GSTP), Minimum Spanning Tree (GMST) and many other geometric problems.

**Applications.** Applications for the minimum corridor connection problem and other generalized geometric problems are naturally encountered in telecommunications, and VLSI design. For instance, a metropolitan area is divided by streets and avenues into rectilinear blocks and the blocks must be interconnected by an optical fiber network containing a gateway from each block. For easy maintenance the optical cables must be placed in the collector system which goes strictly under the streets and avenues. The problem is to find the minimum length network connecting all blocks. In Section 4.5 we discuss how our techniques can be applied to even more generalized variants of this problem. For the related problems and for the extended list of applications see Feremans [10], Feremans, Labbé and Laporte [11], Mitchell [23], Reich and Widmayer [25].

3D-applications of the generalized geometric problems, particularly MCC, appear also in constructions where, e.g., wiring has to be installed along the walls, floors and ceilings of a multistory building such that each room has electricity, phone lines, etc.

**Related Work.** To the best of our knowledge, before this article nothing was known on complexity and approximability of the minimum corridor connection problem; see list of open problems from the 12th Canadian Conference on Computational Geometry CCCG 2000 [6].

For GTSP it is known that the problem cannot be efficiently approximated within $(2-\varepsilon)$ unless P=NP, see [26]. Constant factor approximations for GTSP were developed for the special cases where neighborhoods are disjoint convex fat objects [5,9], for disjoint unit discs [1], and for intersecting unit discs [8]. For the general GTSP, Mata and Mitchell [22] gave an $O(\log n)$-approximation. The closest related work to this article is the paper by Dumitrescu and Mitchell [8], where the authors have investigated the case of GTSP with regions given by pairwise disjoint unit disks, and they developed a polynomial time approximation scheme (PTAS) for this problem.

For the general GSTP, Helvig, Robins, and Zelikovsky [17] developed a polynomial time $n^{\varepsilon}$-approximation algorithm where $\varepsilon > 0$ is any fixed constant. For GSTP, GMST and several other generalized geometric problems exact search methods and heuristics have been developed, see e.g., Zachariasen and Rohe [28], and Feremans, Labbé and Laporte [11].

**Our results and paper organization.** The remainder of this extended abstract is organized as follows. In Section 2 we show that the problem is strongly NP-hard, answering an open question from CCCG 2000 on the complexity of the minimum corridor connection, see [6].

In Section 3 we present a subexponential time exact algorithm for MCC and a cubic time algorithm for the special case when the room connectivity graph is $k$-outerplanar. (We follow [20] for the definition of "subexponential".)

Then, in Section 4 we construct a PTAS for MCC with fat rooms having nearly the same size, that partially solves another open question from CCCG 2000 on the approximability of MCC, see [6]. More precisely, we consider the problem where a square of side length $q$ can be inscribed in each room and the perimeter of each room is bounded from above by $cq$ where $c$ is a constant. In fact, we present a framework for construction the PTASs for a variety of generalized geometric problems restricted to (almost) disjoint fat object of nearly the same size. We refer to this restriction as *geographic clustering* since one can associate disjoint fat objects with countries on a map where all countries have comparable (up to a constant factor) border lengths.

The framework for PTASs presented in this paper is based on Arora's algorithm for ETSP [2]. In particular, this framework allows to construct PTASs for GTSP, GSTP, and GMST restricted to geographic clustering. The main advantage of our techniques compared to the recent approximation scheme by Dumitrescu and Mitchell [8] for GTSP on disjoint unit discs is that it leads to

a more efficient approximation scheme running in time $n(\log n)^{O(1/\varepsilon)}$ compared to $n^{O(1/\varepsilon)}$ in [8]. Moreover, our techniques are applicable to many other norms (e.g., the one which is used in MCC) and to any fixed dimensional spaces, which resolves one of the open questions in [8].

Finally, in Section 5 we show how the algorithm for GTSP from Elbassioni et al. [9] can be used to derive a polynomial time constant approximation algorithm for MCC with fat rooms of varying sizes that complements our partial answer on the open question from CCCG 2000 on the approximability of MCC, see [6].

## 2 Complexity of MCC

In this section, we show that the decision version of MCC is strongly NP-complete. To show this result, we use a transformation from the CONNECTED VERTEX COVER problem for planar graphs with maximum degree four. In this later problem, given a planar graph $G = (V, E)$ such that each vertex in $V$ has degree at most 4, and a positive integer $R \leq |V|$, the question is whether there exists a connected vertex cover of size at most $R$ for $G$, i.e., does there exist a subset $W \subseteq V$ with $|W| \leq R$ such that the subgraph induced by $W$ is connected and for each edge $\{u, v\} \in E$, $u \in W$ or $v \in W$? It is well known that CONNECTED VERTEX COVER for planar graphs with maximum degree four is NP-complete, see [14,15]. Now we state the main result of this section. Because of space constraints, we will omit proofs in this extended abstract.

**Theorem 1.** *The minimum corridor connection problem is NP-complete, even when coordinates of corner points are given in unary.*

## 3 Exact Algorithms with Branchwidth

In this section, we discuss how the problem can be solved exactly exploiting the notion of branchwidth and $k$-outerplanarity.

A *branch decomposition* of a graph $G = (V, E)$ is a pair $(T, \sigma)$, with $T$ an unrooted ternary tree and $\sigma$ a bijection between the leaves of $T$ and the edge set $E$. For each edge $e$ in $T$, consider the two subtrees $T_1$ and $T_2$ obtained by removing $e$ from $T$. Let $G_{e,1}$ ($G_{e,2}$) be the subgraph of $G$, formed by the edges associated with leaves in $T_1$ ($T_2$). The *middle set* of an edge $e$ in $T$ is the set of vertices that are in both $G_{e,1}$ and $G_{e,2}$. The *width* of a branch decomposition is the maximum size over all middle sets, and the *branchwidth* of a graph is the minimum width over all branch decompositions.

A *noose* is a closed simple curve on the plane that intersects the plane graph $G$ only at vertices. To a noose, we can associate two regions of the plane (the "inside" and the "outside"), and likewise two subgraphs: the part of $G$ drawn inside the noose, and the part of $G$ drawn outside the noose. These subgraphs intersect precisely in the vertices on the noose.

A branch decomposition $(T, \sigma)$ is a *sphere cut decomposition* or *sc-decomposition*, if for every edge $e$ in $T$, there is a noose of $G$ such that the two subgraphs

associated with it are exactly $G_{e,1}$ and $G_{e,2}$, and the noose touches each face of $G$ at most once. Necessarily, the set of the vertices on the noose is the middle set of $e$.

A sphere cut decomposition of a plane graph of minimum width can be found in $O(n^3)$ time with the ratcatcher algorithm of Seymour and Thomas [27], see [7]. See also [16,18,19] for a necessary improvements to the original algorithm and implementation issues.

**Dynamic programming with a branch decomposition.** Instead of the MCC problem, we consider a small generalization, which we call FACE COVER TREE: given a plane graph $G = (V, E)$, with edge weights $w : E \to \mathbf{N}$, find a subtree $T$ of $G$ of minimum total weight such that each interior face has at least one vertex on $T$.

We now give an algorithm that solves the FACE COVER TREE problem using a sphere cut decomposition of $G$.

**Theorem 2.** *Suppose a plane graph is given together with a sphere cut decomposition of width at most $k$. Then the* FACE COVER TREE *problem can be solved in $O((3 + \sqrt{5})^k k \cdot n)$ time.*

To obtain this result, we use techniques from Dorn et al. [7]. The basic idea is that we build a table for each edge in the branch decomposition. Assuming a root for $T$, we associate to each edge $e \in E(T)$, the subgraph formed by the edges of $G$ associated with the leaves in $T$ that are below $e$ in the tree. This is one of the subgraphs $G_{e,1}$ or $G_{e,2}$; w.l.o.g., we will assume that this is always $G_{e,1}$. A forest $T'$ that is a subgraph of $G_{e,1}$ can be extended to a solution of the FACE COVER TREE if each face of $G_{e,1}$ that does not intersect the noose is touched by $T'$ and each subtree of $T'$ contains at least one vertex in the middle set of $e$. We can characterize such forests of the second type by the set of vertices in the middle set that belong to the forest, an equivalence relation on these vertices which are connected by the forest, the information which faces that intersect the noose are touched by the forest, and (of course), the total length of all edges in the forest. Having this information is also sufficient to see how the forest can be extended.

Thus, in our dynamic programming algorithm, we tabulate for each edge $e$ in the branch decomposition tree, for each triples $(S, R, X)$, where $S$ is a subset of the middle set of $e$, $R$ is an equivalence relation on $S$, and $X$ is a subset of the faces intersecting the noose of $e$, if there is at least one forest $T'$ in $G_{e,1}$ such that $S$ is the set of vertices in the middle set that belong to $T'$, $R$ is the relation on $S$ that there is a path in $T'$, and $X$ is the set of faces intersecting the noose of $e$ that are touched by $e$, the minimum total weight of such a forest.

Using counting techniques from [7], we can show that for a middle set of size $\ell$, such a table contains at most $(3 + \sqrt{5})^\ell$ entries. (For instance, let $R$ form a non-crossing partition on $S$. We only need to distinguish whether faces are touched whose two incident middle set vertices do not belong to $S$.)

It is trivial to compute the table for an edge in $T$ incident to a leaf. For other edges $e$, we combine the two tables for the two edges incident to the lower endpoint of $e$. Basically, we try to combine each table entry of the left table with

each table entry of the right table; in $O(k)$ time, we can verify whether these give a new table entry, and of what signature. Thus, the table for an edge can be computed in $O((14 + 6\sqrt{5})^k \cdot k)$ time.

From the table of the edge to the root, we can then determine the answer to the problem. We computed $O(n)$ tables, and hence used $O((14 + 6\sqrt{5})^k \cdot k \cdot n)$ time. Note that $14 + 6\sqrt{5} = 2^{4.7770}$.

**Consequences.** Given a plane graph $G = (V, E)$, we can divide the vertices of $G$ into layers. All vertices incident to the exterior face are in layer $L_1$. For $i \geq 1$, all vertices incident to the exterior face after we removed all vertices in layers $L_1, \ldots, L_i$ are in layer $L_{i+1}$. A planar graph $G$ is $k$-outerplanar, if it has a planar embedding with at most $k$ non-empty layers. It is well known that a $k$-outerplanar graph has branchwidth at most $2k$; this can be proved in the same way as the proof in [4] that $k$-outerplanar graphs have treewidth at most $3k - 1$.

It is interesting to note that in some applications, graphs with small outerplanarity will arise in a natural way. For instance, for many buildings, the wall structure of one floor will have bounded outerplanarity, as usually, each room is adjacent to a corridor, and each corridor is adjacent to a room with a window, and thus, unless there is an open air part not at the exterior, this gives small outerplanarity.

It is well long known that planar graphs have branchwidth (and treewidth) $O(\sqrt{n})$. (This statement can be seen to be equivalent to the Lipton-Tarjan planar separator theorem [4,21].) The best known bound to our knowledge is the following.

**Theorem 3 (Fomin and Thilikos [13]).** *A planar graph with $n$ vertices has branchwidth at most $\sqrt{4.5 \cdot n}$.*

Thus we have the following consequences, where we expect that the actual running times of these algorithms will be better in practice.

**Corollary 1.** *The* FACE COVER TREE*, and hence also the MCC problem can be solved in $O(n^3 + 2^{9.5539k})$ time on $k$-outerplanar graphs, and in $O^*(2^{10.1335\sqrt{n}})$ time on planar graphs.*

## 4 A PTAS for MCC with Geographic Clustering

To construct a polynomial time approximation scheme for MCC, we modify Arora's algorithm for ETSP [2,3]. We assume that the corner points of each of the $n$ rooms have integer coordinates, that each room encloses a $q \times q$ square and has perimeter at most $cq$, for some constant $c \geq 4$.

### 4.1 Perturbation and Curved Dissection

Arora's algorithm for ETSP starts with perturbation of the instance that, without great increase of the optimum, ensures that in the resulting new instance all nodes lie on the unit grid, and the maximum internode distance is at most

$poly(n)$. In MCC, perturbation is not necessary. All corner points are already on the integer grid. Further, since all rooms are connected and the perimeter of a room is at most $cq$ the smallest *bounding box* (the smallest axis parallel square containing all rooms) has side length at most $cqn$. Let the size of the bounding box be $L \in [cqn, 2cqn]$ such that $L/cq$ is a power of 2. A simple packing argument shows that the value of the optimal solution is $OPT = \Omega(qn)$.

First we define the *straight dissection* of the bounding box. We stop the partitioning when the side length of the square is $cq$. Since $L \leq 2cqn$ the depth of the dissection tree is $O(\log n)$. Let the *level* of a square be its depth from the root in the straight dissection tree and the *level $i$ dissection lines* are the straight lines participating in the division of the level $i-1$ square into level $i$ sub-squares.

A dissection line can cut a room into two or more parts. This causes troubles for the dynamic programming since we have to determine for each room in which square of the dissection it gets connected. To solve this problem we introduce a *curved dissection*.

Consider a horizontal level dissection line. We replace the line by a dissection curve by walking from left to right and whenever we hit the boundary of a room we follow the boundary (in arbitrary direction) until the dissection line is hit again. The obtained curve may go through some boundary segments twice. We shortcut the curve and obtain a simple path partitioning the set of rooms in an upper and lower set. Vertical dissection curves are defined in a similar way. Moreover, we can easily do this such that each horizontal curve crosses each vertical curve exactly once, i.e., the intersection is one point or a simple path. (See Figures 2 and 3.) Notice that no two horizontal (vertical) dissection curves intersect since, at any point on the curve, the deviation from the dissection line is strictly less than $cq/2$.

The transformation of lines to curves maps each node of the straight dissection tree onto a polygon which we denote by *node polygons* of the *curved dissection* tree of the bounding box.

In Figure 2, dissection lines are depicted by dotted lines and dissection curves are depicted by fat piece-wise linear curves. Notice that the middle room is crossed by vertical and horizontal dissection lines.

### 4.2 Portals and Portal Respecting Trees

Let a level $i$ dissection curve have $2^i m$ special points equally spaced on that curve. By equally spaced we mean that the piece-wise linear fragments of the

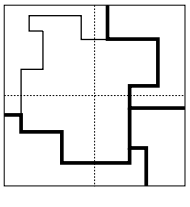


**Fig. 2.** Curved dissection

curve between two consecutive points have the same length. We refer to these points as *portals* and to $m$ as *portal parameter* (to be defined later).

Remember that the intersection of a horizontal and vertical curve is in general a path. The definition above leads to two sets of portals on such paths. We keep only the portals of the highest level curve and pick one set arbitrarily if levels are equal. Further, we define one portal on both endpoints of each path of intersection which we call *corner portals*.

To make the dynamic programming work we have to assume that if some segment of the tree coincides with a dissection curve, it can only connect rooms on one side of the curve. To serve rooms at the other side it has to cross the curve. (See Figure 3.) We call a feasible tree *portal respecting* if these crossings only appear at portals. We refer to the boundary segment of the node polygon belonging to the dissection curve as the *side* of the node polygon. Notice that sides may overlap. A portal respecting tree is *k-light* if it crosses each side of each node polygon at most $k$ times.

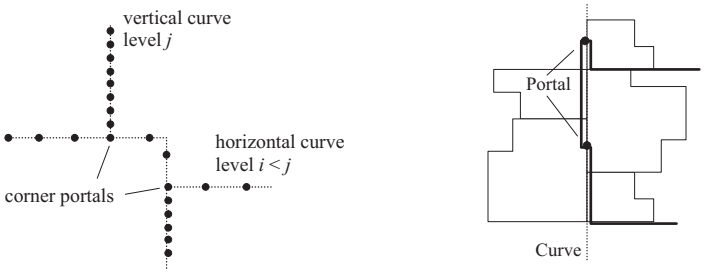



**Fig. 3.** Portals and a feasible portal respecting tree

### 4.3 The Algorithm

First we construct the bounding box with the dissection curves. Since each room is adjacent to at most two curves the construction can be done in $O(n)$ time. Next we choose $a, b \in \{1, 2, \ldots, L/(cq)\}$ at random and make the $a$-th horizontal and $b$-th vertical dissection curve the level zero curves. The curved dissection tree is now build in a wrap-around manner as in Arora [3]. By removing from the 4-ary tree all branches consisting of empty node polygons, we obtain a tree having at most $O(n)$ leaves and $O(n \log n)$ node polygons. Then we define the portals as in Section 4.2. Starting at the leaves of the dissection tree in a bottom-up way, we update the dynamic programming table. For each node polygon, for each $k$-elementary subset of the portals on the boundary of the polygon, and for each partition $B_1, \ldots, B_p$ of these $k$ portals, we store the length of the optimal forest consisting of $p$ trees which together touch all rooms and the $i$-th tree connects all portals in $B_i$.

For the node polygons in the leaves of the dissection tree we simply enumerate all such forests, since these polygons contain at most $c^2$ rooms. For the root polygon we guess the information for the portals on the two level one dissection curves separating the root polygon. We make sure that the four forests

together form one tree. The number of different problems in one node polygon is $O(m^{O(k)} f(k))$ for some function $f$. Taking $m = O(\frac{\log n}{\varepsilon})$ and $k = O(\frac{1}{\varepsilon})$ the size of the look up table is $O(n \log^{\gamma} n)$, for some constant $\gamma$.

### 4.4 Performance Guarantee

The performance guarantee follows from the following theorem.

**Theorem 4 (Structure Theorem).** *Let $OPT_{a,b,k,m}$ be the length of the minimum $k$-light portal respecting tree when the portal parameter is $m$.*

$$E[OPT_{a,b,k,m} - OPT] \leq \left( O\left(\frac{\log n}{m}\right) + O\left(\frac{1}{k-4}\right)\right) OPT,$$

*where $E[\cdot]$ is over the random choice of $(a, b)$-shift.*

The proof is omitted and is slightly more complicated than in Arora [3]. Taking $m = O(\frac{\log n}{\varepsilon})$ and $k = O(\frac{1}{\varepsilon})$ we derive the following result.

**Theorem 5.** *The randomized algorithm described above returns a feasible tree of length at most $(1 + \varepsilon)OPT$ in time $n(\log n)^{O(1/\varepsilon)}$.*

To derandomize the algorithm, we can simply go through all possible choices for $a$ and $b$. More sophisticated derandomization techniques are described in Rao and Smith [24]. In fact, a straightforward adaption of a more careful analysis presented in [24] can also significantly improve the running times presented in this extended abstract. For two dimensional space this would even imply an $O(n \log n)$ time and $O(n)$ space PTAS for MCC and other geometric problems with geographic clustering.

### 4.5 Extensions of the PTAS

As in Arora [2,3] we did not use much of the specifics of MCC. The basic idea to tackle the generalized geometric problems with geographic clustering is to introduce the curved dissection, new stoppage criteria and then to use the fact that under geographic clustering the lengths of the dissection curves only differ by a constant factor from the lengths of the dissection lines, yielding the same (up to a constant factor) charges to the objective function as in non-generalized versions of the geometric problems. In this way, with slight modifications in the analysis of the algorithm, we can derive PTASs for GTSP, GSTP, GMST and many other generalized geometric problems. Moreover, the approach is naturally applicable to many other norms, e.g., we can straightforwardly adopt the approximation scheme to any $L_p$ norm. Also notice, that the requirement that the partition of the polygon must be rectilinear is not crucial. It is sufficient to assume that the walls of each room are given by a sequence of line segments forming a simple closed walk in the plane (here, the only critical assumption is that all rooms must be fat and have comparable sizes, i.e., for each room its perimeter must be bounded by $cq$ where $q$ is the minimum size over all rooms of the maximum inscribed square or ball and $c$ is a fixed constant).

Dumitrescu and Mitchell in [8] pointed out that in their approximation scheme for GTSP only some of the arguments for disjoint discs can be lifted to higher dimensions and, naturally, one of the open questions they listed was: "What approximation bounds can be obtained in higher dimensions?" It is well known, see e.g., [2,3,24], that Arora's algorithm for ETSP is applicable also in higher fixed dimensional spaces. Using literally the same argumentation as in [2] and our construction for MCC with geographic clustering, one can derive the following theorem.

**Theorem 6.** *If the corner points of the rooms are in $\mathcal{R}^d$, the MCC with geographic clustering admits a randomized PTAS running in $n(\log n)^{(O(\sqrt{d}/\varepsilon))^{d-1}}$ time. Derandomization of the algorithm in this case will cost an additional factor of $O(n^d)$ leading to overall running time of $n^{d+1}\ (\log n)^{(O(\sqrt{d}/\varepsilon))^{d-1}}$.*

The same holds for GTSP, GSTP and GMST. This resolves the open question from Dumitrescu and Mitchell [8].

## 5 An Approximation Algorithm for MCC with Rooms of Varying Sizes

Elbassioni et al. [9] give a simple constant factor approximation algorithm for GTSP, where the factor depends on the fatness of the regions. Here we modify their algorithm and proof to obtain a constant factor approximation algorithm for MCC.

For any room $R_i$, $i \in \{1, \ldots, n\}$, we define its *size* $\rho_i$ as the side length of the smallest enclosing square of the room. We restrict to rooms for which the perimeter is bounded by the size of the room, lets say at most $4\rho_i$. A room $R$ is said to be $\alpha$-*fat* if for any square $Q$ whose boundary intersects $R$ and whose center lies in $R$, the area of the intersection of $R$ and $Q$ is at least $\alpha/4$ times the area of $Q$. Note that the fatness of a square is 1 and in general $\alpha \in [0, 1]$.
Algorithm Greedy:

(1) Pick the corner points $p_i \in R_i$, $i \in \{1, \ldots, n\}$, that minimize $\sum_{i=2}^{n} d(p_1, p_i)$, where $d(x, y)$ is the shortest distance between $x$ and $y$ along the walls.
(2) Let $G$ be a graph with a vertex $v_i$ for every room $R_i$ and $d(v_i, v_j) = d(p_i, p_j)$. Find a minimum spanning tree $T$ in $G$.
(3) Construct a solution to MCC as follows. For every edge $(v_i, v_j)$ in $T$, let the minimum length $(p_i, p_j)$-path belong to the corridor. If the resulting corridor is not a tree, break the cycles (removing edges) arbitrarily.

**Lemma 1.** *Algorithm* Greedy *gives an $(n-1)$-approximate solution for MCC.*

*Proof.* Consider an optimal solution and let $OPT$ be its length. Identify for each room $R_i$ a point $p'_i$ in the room that is connected to the optimal tree. The optimal tree contains a path from $p'_1$ to $p'_i$ for all $i \in \{2, \ldots, n\}$. Therefore, $(n-1)OPT \geq \sum_{i=2}^{n} d(p'_1, p'_i) \geq \sum_{i=2}^{n} d(p_1, p_i)$, which is at most the length of the tree constructed by the algorithm. □

**Lemma 2.** *The length of the shortest corridor that connects $k$ rooms is at least $\rho_{\min}(k\alpha/2-2)$, where $\rho_{\min}$ is the size of the smallest of these rooms.*

*Proof.* Let $P$ be a connecting corridor and let $d(P)$ denote its length (along the walls). Let the center of a square with side length $2\rho_{\min}$ move along the corridor $P$. The total area $A$ covered by the moving square is at most $(2\rho_{\min})^2 + 2\rho_{\min} \cdot d(P)$. Assume a room is connected with $P$ at point $p$. Putting the center of the square in point $p$ we see that its boundary intersects the room. By definition of $\alpha$ at least a fraction $\alpha/4$ of the room is contained in the square. Therefore, $k(2\rho_{\min})^2\alpha/4$ is a lower bound on the area $A$. We have $k(2\rho_{\min})^2\alpha/4 \leq A \leq (2\rho_{\min})^2 + 2\rho_{\min} \cdot d(P)$, yielding $d(P) \geq \rho_{\min}(k\alpha/2-2)$, which completes the proof. □

Algorithm CONNECT:

(1) Order the rooms by their sizes $\rho_1 \leq \rho_2 \leq \ldots \leq \rho_n$. Pick any $p_1$ on the boundary of $R_1$. For $i = 2$ up to $n$ pick the point $p_i$ in $R_i$ that minimizes $\min\{d(p_i, p_1), d(p_i, p_2), \ldots, d(p_i, p_{i-1})\}$, i.e., pick the point that is closest to the already chosen points.
(2) Let $G$ be a graph with a vertex $v_i$ for every room $R_i$ and $d(v_i, v_j) = d(p_i, p_j)$. Find a minimum spanning tree $T$ in $G$.
(3) Construct a solution to MCC as follows. For every edge $(v_i, v_j)$ in $T$, let the minimum length $(p_i, p_j)$-path belongs to the corridor. If the resulting corridor is not a tree, break the cycles (removing edges) arbitrarily. Output the minimum of the obtained tree and the tree constructed by algorithm GREEDY.

**Theorem 7.** *Algorithm* CONNECT *gives a $(16/\alpha - 1)$-approximate solution for the minimum corridor connection problem in which the fatness of every room is at least $\alpha$.*

*Proof.* If $n - 1 \leq 16/\alpha - 1$ then GREEDY guarantees the approximation ratio for smaller values of $n$. So assume $n \geq 16/\alpha$. Denote the set of points chosen by CONNECT as $P' = \{p_1, \ldots, p_n\}$. Let $p_i^*$ be the point from $\{p_1, \ldots, p_{i-1}\}$ that is at minimum distance from $p_i$. Denote the distance $d(p_i, p_i^*)$ by $x_i$.

Consider some *closed* walk $\Omega$ connecting all rooms and assume its length is minimum. The length of this walk is clearly an upper bound on $OPT$. For each room $R_i$, $i \in \{1, \ldots, n\}$, we define one connection point $r_i$ on $\Omega$ in which it hits the room. Consider one of the two possible directions of $\Omega$ and assume that the tour connects the rooms in the order $1, 2, \ldots, n$. Let $k \in \{1, \ldots, n\}$. We define $T_i$ as the part of this directed walk that connects exactly $k$ rooms at their connection points and starts from point $r_i$. Let $t_i$ be the length of the (not necessarily simple) path $T_i$. We have $OPT \leq d(\Omega) = \sum_{i=1}^{n} t_i/(k-1)$.

Consider some $i \in \{1, \ldots, n\}$ and let $R_{h(i)}$ be the smallest room among those from the $k$ rooms on the path $T_i$. Since $R_i$ is on this path $T_i$ and we ordered the rooms by their size we may assume $1 \leq h(i) \leq i$. We partition the rooms into two sets. Let $F$ be the set of rooms for which $h(i) = i$ and let $H$ contain the remaining rooms. Let $T'$ be an MST on the point set $P'$ restricted to the rooms

in $F$. Then $d(T') \leq OPT + 2\sum_{i \in F} \rho_i$. The connected graph that we construct consists of the edges of $T'$ and for all rooms $i$ in $H$ we add the path $(p_i, p_i^*)$ which has length $x_i$. Note that the resulting graph is indeed connected and has total length at most

$$OPT + \sum_{i \in F} 2\rho_i + \sum_{i \in H} x_i.$$

We define $\gamma = k\alpha/2 - 2$. From Lemma 2 we know

$$t_i \geq \gamma \rho_i, \text{ for all } i \in F. \tag{1}$$

If $i \in H$, then we argue as follows. Since the algorithm picked point $p_i$ we know that the distance from any point in $R_i$ to the point $p_{h(i)}$ (which is chosen before $p_i$) is at least $x_i$. Hence, the distance from any point in $R_i$ to any point in $R_{h(i)}$ is at least $x_i - 2\rho_{h(i)}$, implying $t_i \geq x_i - 2\rho_{h(i)}$. Additionally, we know from Lemma 2 that $t_i \geq \gamma \rho_{h(i)}$. Combining the two bounds we get

$$t_i \geq \max\{\gamma \rho_{h(i)}, x_i - 2\rho_{h(i)}\} \geq \frac{\gamma}{\gamma + 2} x_i, \text{ for all } i \in H. \tag{2}$$

Combining (1) and (2) we see that the MST given by the algorithm has length at most

$$\begin{aligned} OPT + \sum_{i \in F} 2/\gamma t_i + \sum_{i \in H} (1 + 2/\gamma) t_i &\leq OPT + \sum_{i=1}^{n} (1 + 2/\gamma) t_i \\ &\leq OPT + (1 + 2/\gamma)(k-1)OPT \\ &= OPT + (1 + 2/(k\alpha/2 - 2))(k-1)OPT \\ &= OPT + \tfrac{k(k-1)}{k - 4/\alpha} OPT \end{aligned}$$

It is easy to show that $k(k-1)/(k - 4/\alpha)$ equals $16/\alpha - 2$ for $k = 8/\alpha - 1$ and also for $k = 8/\alpha$. Furthermore, it is strictly smaller for any value in between. Hence, there is an integer $k \in [8/\alpha - 1, 8/\alpha]$ such that $k(k-1)/(k - 4/\alpha) \leq 16/\alpha - 2$. Note that by the assumption in the first line of the proof we satisfy $k \in \{1, \ldots, n\}$. We conclude that the length of the tree given by the algorithm is at most $(16/\alpha - 1)OPT$. □

## Acknowledgments

We thank Joe Mitchell, Sándor Fekete, and Mark de Berg for useful discussions on the GTSP with geographic clustering, and also thanks to Sergio Cabello.

## References

1. E. M. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55:197–218, 1994.
2. S. Arora. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. *Journal of the ACM*, 45:1–30, 1998.

3. S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Mathematical Programming*, 97:43–69, 2003.
4. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.
5. M. de Berg, J. Gudmundsson, M. Katz, C. Levcopoulos, M. Overmars, and A. van der Stappen. TSP with neighborhoods of varying size. *Journal of Algorithms*, 57:22–36, 2005.
6. E. D. Demaine and J. O'Rourke. Open problems from CCCG 2000. `http://theory.lcs.mit.edu/~edemaine/papers/CCCG2000Open/`, 2000.
7. F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. In *Algorithms - ESA 2005, 13th Annual European Symposium*, pages 95–106. LNCS 3669, Springer, October 2005.
8. A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *Journal of Algorithms*, 48:135–159, 2003.
9. K. Elbassioni, A. V. Fishkin, N. H. Mustafa, and R. Sitters. Approximation algorithms for Euclidean group TSP. In *Automata, Languages and Programming: 32nd International Colloquium, ICALP*, pages 1115–1126. LNCS 3580, Springer, July 2005.
10. C. Feremans. *Generalized Spanning Trees and Extensions*. PhD thesis, Université Libre de Bruxelles, Brussels, 2001.
11. C. Feremans, M. Labbé, and G. Laporte. Generalized network design problems. *European Journal of Operational Research*, 148:1–13, 2003.
12. M. Fischetti, J. J. Salazar, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45:378–394, 1997.
13. F. V. Fomin and D. M. Thilikos. New upper bounds on the decomposability of planar graphs. *Journal of Graph Theory*, 51:53–81, 2006.
14. M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.
15. M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.
16. Q. Gu and H. Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005*, pages 373–384. LNCS 3580, Springer, July 2005.
17. C. S. Helvig, G. Robins, and A. Zelikovsky. An improved approximation scheme for the Group Steiner Problem. *Networks*, 37:8–20, 2001.
18. I. V. Hicks. Planar branch decompositions I: The ratcatcher. *INFORMS Journal on Computing*, 17:402–412, 2005.
19. I. V. Hicks. Planar branch decompositions II: The cycle method. *INFORMS Journal on Computing*, 17:413–421, 2005.
20. R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
21. R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math*, 36:177–189, 1979.
22. C. S. Mata and J. S. B. Mitchell. Approximation algorithms for geometric tour and network design problems. In *ACM SoCG 1995: Vancouver, BC, Canada*, pages 360–369. ACM, 1995.
23. J. S. B. Mitchell. *Handbook of Computational Geometry*, chapter Geometric shortest paths and network optimization, pages 633–701. Elsevier, North-Holland, Amsterdam, 2000.

24. S. Rao and W. D. Smith. Approximating geometric graphs via "spanners" and "banyans". In *ACM STOC 1998: Dallas, Texas, USA*, pages 540–550. ACM, 1998.
25. G. Reich and P. Widmayer. Beyond Steiner's problem: a VLSI oriented generalization. In *Graph-Theoretic Concepts in Computer Science, 15th International Workshop, WG '89*, pages 196–210. LNCS 411, Springer, 1990.
26. S. Safra and O. Schwartz. On the complexity of approximating TSP with neighborhoods and related problems. In *Algorithms - ESA 2003, 11th Annual European Symposium*, pages 446–458. LNCS 2832, Springer, September 2003.
27. P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14:217–241, 1994.
28. M. Zachariasen and A. Rohe. Rectilinear group Steiner trees and applications in VLSI design. *Mathematical Programming*, 94:407–433, 2003.

# Online $k$-Server Routing Problems

Vincenzo Bonifaci[1,2,⋆] and Leen Stougie[1,3,⋆⋆]

[1] Department of Mathematics and Computer Science
Eindhoven University of Technology
Den Dolech 2 – PO Box 513, 5600 MB Eindhoven, The Netherlands
v.bonifaci@tue.nl, l.stougie@tue.nl
[2] Department of Computer and Systems Science
University of Rome "La Sapienza"
Via Salaria, 113 – 00198 Roma, Italy
bonifaci@dis.uniroma1.it
[3] CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
stougie@cwi.nl

**Abstract.** In an online $k$-server routing problem, a crew of $k$ servers has to visit points in a metric space as they arrive in real time. Possible objective functions include minimizing the makespan ($k$-Traveling Salesman Problem) and minimizing the average completion time ($k$-Traveling Repairman Problem). We give competitive algorithms, resource augmentation results and lower bounds for $k$-server routing problems on several classes of metric spaces. Surprisingly, in some cases the competitive ratio is dramatically better than that of the corresponding single server problem. Namely, we give a $1 + O((\log k)/k)$-competitive algorithm for the $k$-Traveling Salesman Problem and the $k$-Traveling Repairman Problem when the underlying metric space is the real line. We also prove that similar results cannot hold for the Euclidean plane.

## 1 Introduction

In a $k$-server routing problem, $k$ servers (vehicles) move in a metric space in order to visit a set of points (cities). Given a schedule, that is, a sequence of movements of the servers, the time at which a city is visited for the first time by one of the servers is called the *completion time* of the city. The objective is to find a schedule that minimizes some function of the completion times.

We study $k$-server routing problems in their *online* version, where decisions have to be taken without having any information about future requests. New requests may arrive while processing previous ones. This online model is often called the *real time* model, in contrast to the *one-by-one* model, which is the more common model in texts about online optimization [5], but inadequate for server routing problems. The same real time model is also the natural model and indeed

⋆ Partly supported by the Dutch Ministry of Education, Culture and Science through a Huygens scholarship.
⋆⋆ Partly supported by MRT Network ADONET of the European Community (MRTN-CT-2003-504438) and the Dutch BSIK/BRICKS project.

T. Erlebach and C. Kaklamanis (Eds.): WAOA 2006, LNCS 4368, pp. 83–94, 2006.

is used for machine scheduling problems [22]. In fact, many of the algorithms for online routing problems are adaptations of online machine scheduling algorithms.

*Competitive analysis* [5] has become the standard way to study online optimization problems: an online algorithm $A$ is said to be *$c$-competitive* if, for any instance $\sigma$, the cost of $A$ on $\sigma$ is at most $c$ times the offline optimum cost on $\sigma$. This worst-case measure can be seen as the outcome of a game between the online algorithm and an offline *adversary*, that is trying to build input instances for which the cost ratio is as large as possible.

There is an abundant amount of literature on offline server routing problems, both in past and recent times [7, 10, 12, 14, 18]. Online single server routing problems have a recent but growing literature. The first paper by Ausiello et al. [3] introduced the model for the online traveling salesman problem. Later works investigated competitiveness of the more general dial-a-ride problems [1, 11] and studied different objective functions or different adversarial models [2, 4, 13, 16, 17, 20]. A summary of single server results is contained in the thesis [19].

Prior to this publication, there was essentially no work on online multi-server routing problems, except for some isolated algorithms [1, 4]. We give competitive algorithms and negative results for online multi-server routing problems, with the objective of minimizing either *makespan* or *average completion time*. In the case of makespan we consider the variant known as *nomadic*, in which the servers are not required to return at the origin after serving all requests; the above cited previous results refer to the other variant, known as the *homing* traveling salesman problem. Apart from being the first paper dedicated to multi-server online routing problems, the results are somewhat unexpected. We give the first results of online problems for which multiple server versions admit lower competitive ratios than their single server counterparts. This is typically not the case for problems in the one-by-one model; for example, it is known that in the famous *$k$-server* problem [21] the competitive ratio necessarily grows linearly with $k$.

It may also be useful to draw a comparison with machine scheduling, which is closer to routing problems in many ways. In scheduling a lot of research has been conducted to online multiple machine problems [22]. In the one-by-one model competitive ratios increase with increasing number of machines. In real time online scheduling nobody has been able to show smaller competitive ratios for multiple machine problems than for the single machine versions, though here lower bounds do not exclude that such results exist (and indeed people suspect they do) [8, 9].

The rest of our paper is structured as follows. After introducing our model in Section 2, we give in Section 3 competitive algorithms and lower bounds for both the $k$-Traveling Salesman and the $k$-Traveling Repairman in general spaces. For these algorithms, the upper bounds on the competitive ratio match those of the best known algorithms for the single server versions. In Section 4, we show that in the case of the real line we have an almost optimal algorithm for large $k$. The same result cannot hold in the Euclidean plane, as we show in Section 5. We give our conclusions in Section 6.

## 2 Preliminaries

We assume a real time online model, in which requests arrive over time in a metric space $\mathbb{M}$. Every *request* is a pair $(r, x) \in \mathbb{R}_+ \times \mathbb{M}$ where $r$ is the *release date* of the request and $x$ the location of the request. All the information about a request with release date $r$, including its existence, is revealed only at time $r$. Thus, an online algorithm does not know when all requests have been released.

An algorithm controls $k$ vehicles or *servers*. Initially, at time 0, all these servers are located in a distinguished point $o \in \mathbb{M}$, the origin. The algorithm can then move the servers around the space at speed at most 1. (We do not consider the case in which servers have different maximum speeds; in compliance with machine scheduling vocabulary we could say that the servers are identical and work in parallel.) To process, or *serve*, a request, a server has to visit the associated location, but not earlier than the release date of the request.

We consider so-called *path metric* spaces, in which the distance $d$ between two points is equal to the length of the shortest path between them. We also require the spaces to be continuous, in the sense that $\forall x, y \in \mathbb{M}\ \forall a \in [0, 1]$ there is $z \in \mathbb{M}$ such that $d(x, z) = ad(x, y)$ and $d(z, y) = (1 - a)d(x, y)$. A discrete space, like a weighted graph, can be extended to a continuous path metric space in the natural way; the continuous space thus obtained is said to be *induced* by the original space. We recall that a function $d : \mathbb{M}^2 \to \mathbb{R}_+$ is a *metric* if satisfies: definiteness ($\forall x, y \in \mathbb{M}$, $d(x, y) = 0 \Leftrightarrow x = y$); symmetry ($\forall x, y \in \mathbb{M}$, $d(x, y) = d(y, x)$); triangle inequality ($\forall x, y, z \in \mathbb{M}$, $d(x, z) + d(z, y) \geq d(x, y)$). When referring to a *general space*, we mean any element of our class of continuous, path metric spaces. We will also be interested in special cases, namely the real line $\mathbb{R}$ and the real halfline $\mathbb{R}_+$, both with the origin $o$ at 0, and the plane $\mathbb{R}^2$, with $o$ at $(0, 0)$.

Defining the *completion time* of a request as the time at which the request has been served, the *$k$-traveling salesman problem* ($k$-TSP) has objective minimizing the *maximum completion time*, the *makespan*, and the *$k$-traveling repairman problem* ($k$-TRP) has objective minimizing the *average completion time*.

We will use $\sigma$ to denote a sequence of requests. Given $\sigma$, a feasible *schedule* for $\sigma$ is a sequence of moves of the servers such that all requests in $\sigma$ are served. $\textsc{ol}(\sigma)$ is the cost online algorithm OL incurs on $\sigma$, and $\textsc{opt}(\sigma)$ the optimal offline cost on $\sigma$. OL is said to be $c$-competitive if $\forall \sigma\ \textsc{ol}(\sigma) \leq c \cdot \textsc{opt}(\sigma)$.

We use $s_1, \ldots, s_k$ to denote the $k$ servers, and write $s_j(t)$ for the position of server $s_j$ at time $t$, and $d_j(t)$ for $d(s_j(t), o)$. Finally, given a path $P$ in $\mathbb{M}$, we denote its length by $|P|$.

All the lower bounds we prove hold for randomized algorithms against an oblivious adversary [5]. In order to prove these results, we frequently resort to the following form of Yao's principle [6, 23].

**Theorem 2.1 (Yao's principle).** *Let $\{\textsc{ol}_y : y \in \mathcal{Y}\}$ denote the set of deterministic online algorithms for an online minimization problem. If $X$ is a distribution over input sequences $\{\sigma_x : x \in \mathcal{X}\}$ such that*

$$\inf_{y \in \mathcal{Y}} \mathbb{E}_X[\textsc{ol}_y(\sigma_x)] \geq c\, \mathbb{E}_X[\textsc{opt}(\sigma_x)]$$

**Algorithm 1.** Group Return Home (GRH)

Divide the servers into $g = \lfloor k/k^* \rfloor$ disjoint sets (*groups*) of $k^*$ servers each. Any remaining server is not used by the algorithm.
Initially, all servers wait at $o$. Every time a new request arrives, all servers not at $o$ return to the origin at full speed. Once all of the servers in one of the groups, say group $G$ (ties broken arbitrarily), are at $o$, compute a set of $k^*$ paths $\{P_1, \ldots, P_{k^*}\}$ starting at $o$, covering all unserved requests and minimizing $\max_i |P_i|$. Then, for $i = 1, \ldots, k^*$, the $i$-th server in $G$ follows path $P_i$ at the highest possible speed while remaining at a distance at most $\alpha t$ from $o$ at any time $t$, for some constant $\alpha \in (0, 1]$. Servers in other groups continue to head towards $o$ (or wait there) until a new request is released.

*for some real number $c \geq 1$, then $c$ is a lower bound on the competitive ratio of any randomized algorithm against an oblivious adversary.*

## 3 Algorithms for General Metric Spaces

In this section, we give competitive algorithms and lower bounds for the $k$-TSP and the $k$-TRP in general spaces. Our results will be formulated in a more general *resource augmentation* framework [15]. We define the $(k, k^*)$-TSP and $(k, k^*)$-TRP exactly as the $k$-TSP and the $k$-TRP, except that we measure the performance of an online algorithm with $k$ servers relative to an optimal offline algorithm with $k^* \leq k$ servers. Throughout the section, we let $g = \lfloor k/k^* \rfloor$.

Sections 3.1 and 3.2 give an algorithm for the $(k, k^*)$-TSP and the $(k, k^*)$-TRP respectively. A lower bound for both problems is proved in Section 3.3.

### 3.1 The $k$-Traveling Salesman Problem

**Theorem 3.1.** *There is a deterministic online algorithm for the $(k, k^*)$-TSP with competitive ratio*

$$1 + \sqrt{1 + 1/2^{\lfloor k/k^* \rfloor - 1}}.$$

The algorithm achieving this bound is called Group Return Home (Algorithm 1). Define the *distance of a group to the origin* at time $t$ as the maximum distance of a server in the group to $o$ at time $t$.

**Lemma 3.1.** *At any time $t$, in the schedule generated by GRH, let $G_1(t), \ldots, G_g(t)$ be the $g$ groups in order of nondecreasing distance to $o$. Then the distance of $G_i(t)$ to $o$ is at most $2^{i-g}\alpha t$.*

*Proof.* We prove the lemma by induction on the number of requests. That is, we show that if the lemma holds at the release date $t$ of some request, it will hold until the release date $t + \delta$ of the next request. Obviously, the lemma is true up to the time the first request is given, since all servers remain at $o$.

Suppose a request is given at time $t$. By induction, we know that there are groups $G_1(t), \ldots, G_g(t)$ such that each server of group $G_i(t)$ is at distance at most

$2^{i-g}\alpha t$ from $o$. For the rest of the proof we fix the order of the groups as the order they have at time $t$ and write $G_i$ instead of $G_i(t)$. Let $D_i(\tau) = \max_{s \in G_i} d(s(\tau), o)$.

Between time $t$ and $t' = t + D_1(t)$, the lemma holds since all servers are getting closer to $o$. We show that the lemma holds at $t' + \delta$ for all $\delta > 0$. Notice that $D_1(t' + \delta) \leq \delta$ since every server moves at most at unit speed.

If $\delta \in (0, 2^{1-g}\alpha t]$, we know that $D_1(t' + \delta) \leq 2^{1-g}\alpha t$, so the lemma holds with the groups in the same order as before.

Now, let $\delta \in (2^{i-1-g}\alpha t, 2^{i-g}\alpha t]$ for $2 \leq i \leq g$. Then at time $t' + \delta$, group $G_j$ is already at $o$ for each $1 < j < i$. For group $G_i$, $D_i(t' + \delta) \leq 2^{i-g}\alpha t - 2^{i-1-g}\alpha t = 2^{i-1-g}\alpha t$. For group $G_1$, $D_1(t' + \delta) \leq 2^{i-g}\alpha t$. For groups $G_{i+1}$ through $G_g$, $D_{i+1}(t' + \delta) \leq 2^{i+1-g}\alpha t, \ldots, D_g(t' + \delta) \leq 2^0 \alpha t$. So the lemma holds for these values of $\delta$.

The last case is $\delta > \alpha t$. In this case all groups except $G_1$ are at $o$, and because of the speed constraint $D_1(t' + \delta) \leq \alpha(t' + \delta)$. Thus the lemma holds. □

*Proof (of Theorem 3.1).* Let $t$ be the release date of the last request and let $G_1$ be the group minimizing the distance to the origin at time $t$. Using Lemma 3.1 we know that $D_1(t) \leq 2^{1-g}\alpha t$. Group $G_1$ will return to the origin and then follow the offline set of paths $\{P_1, \ldots, P_{k^*}\}$. Notice that $\text{OPT}(\sigma) \geq t$, since no schedule can end before the release date of a request, and $\text{OPT}(\sigma) \geq \max_i |P_i|$ because of the optimality of the $P_i$.

Let $s$ be the server in $G_1$ that achieves the makespan. If $s$ does not limit its speed after time $t$, we have $\text{OL}(\sigma) \leq t + D_1(t) + \max_i |P_i| \leq (2 + 2^{1-g}\alpha)\text{OPT}(\sigma)$.

Otherwise, let $t'$ be the last time at which $s$ is moving at limited speed. It is not difficult to see that $s$ must serve some request at that time. Let $x_0$ be the location of this request. Then $t' = (1/\alpha)d(x_0, o)$ and $s$ continues following the remaining part of its path, call it $P'$, at full speed. Hence, $\text{OL}(\sigma) = t' + |P'|$. Since $\text{OPT}(\sigma) \geq \max_i |P_i| \geq d(o, x_0) + |P'|$ this yields $\text{OL}(\sigma) \leq (1/\alpha)\text{OPT}(\sigma)$.

Thus, the competitive ratio is at most $\max\{2 + 2^{1-g}\alpha, 1/\alpha\}$ and choosing $\alpha$ in order to minimize it gives $\alpha = \sqrt{2^{g-1}(2^{g-1} + 1)} - 2^{g-1}$ and the desired competitive ratio. □

**Corollary 3.1.** *There is a deterministic* $(1 + \sqrt{2})$*-competitive online algorithm for the* $k$*-TSP.*

### 3.2 The $k$-Traveling Repairman Problem

**Theorem 3.2.** *There is a deterministic online algorithm for the* $(k, k^*)$*-TRP with competitive ratio* $2 \cdot 3^{1/\lfloor k/k^* \rfloor}$.

We call the algorithm achieving the bound Group Interval (Algorithm 2), as it can be seen as a multi-server generalization of algorithm Interval [16]. The algorithm is well defined since the time between two departures of the same group is enough for the group to complete its first schedule and return to the origin: $B_{i+g} - B_i = 2B_i$.

To sketch the proof of Theorem 3.2, we start with two auxiliary lemmas.

**Algorithm 2.** Group Interval (GI)

Divide the servers into $g = \lfloor k/k^* \rfloor$ disjoint sets (groups) of $k^*$ servers each. Any remaining server is not used by the algorithm.
Let $L$ be the earliest time that any request can be completed (wlog $L > 0$). For $i = 0, 1, \ldots$, define $B_i = \alpha^i L$ where $\alpha = 3^{1/g}$.
At time $B_i$, compute a set of paths $S_i = \{P_1^i, \ldots, P_{k^*}^i\}$ for the set of yet unserved requests released up to time $B_i$ with the following properties:
(i) every $P_j^i$ starts at the origin $o$;
(ii) $\max_j |P_j^i| \leq B_i$;
(iii) $S_i$ maximizes the number of requests served among all schedules satisfying the first two conditions.
Starting at time $B_i$, the $j$-th server in the ($i$ mod $g$)-th group follows path $P_j^i$, then returns to $o$ at full speed.

**Lemma 3.2 ([16]).** *Let $a_i, b_i \in \mathbb{R}$ for $i = 1, \ldots, p$, for which*

*(i)* $\sum_{i=1}^{p} a_i = \sum_{i=1}^{p} b_i$, *and*
*(ii)* $\sum_{i=1}^{p'} a_i \geq \sum_{i=1}^{p'} b_i$ *for all* $1 \leq p' \leq p$.

*Then the* $\sum_{i=1}^{p} \tau_i a_i \leq \sum_{i=1}^{p} \tau_i b_i$ *for any nondecreasing sequence of real numbers* $0 \leq \tau_1 \leq \tau_2 \leq \ldots \leq \tau_p$.

**Lemma 3.3.** *Let $R_i$ be the set of requests served by the set of paths $S_i$ computed by Group Interval at time $B_i$, $i = 1, 2, \ldots$ and let $R_i^*$ be the set of requests in the optimal offline solution that are completed in the time interval $(B_{i-1}, B_i]$. Then*

$$\sum_{i=1}^{q} |R_i| \geq \sum_{i=1}^{q} |R_i^*| \quad \text{for all } q = 1, 2, \ldots.$$

*Proof.* We omit the proof, as it is basically the same as that of Lemma 4 in [16]. □

*Proof (of Theorem 3.2).* Let $\sigma = \sigma_1 \ldots \sigma_m$ be any sequence of requests. By construction of Group Interval, each request in $R_i$ is served at most at time $2B_i$. Now, let $p$ be such that the optimal offline schedule completes in the interval $(B_{p-1}, B_p]$. Summing over all phases $1, \ldots, p$ yields

$$\text{OL}(\sigma) \leq 2 \sum_{i=1}^{p} B_i |R_i| = 2 \cdot 3^{1/g} \sum_{i=1}^{p} B_{i-1} |R_i|. \tag{1}$$

From Lemma 3.3 we know that $\sum_{i=1}^{q} |R_i| \geq \sum_{i=1}^{q} |R_i^*|$ for $q = 1, 2, \ldots$ We also know that $\sum_{i=1}^{p} |R_i| = \sum_{i=1}^{p} |R_i^*|$. Applying Lemma 3.2 to the sequences $a_i := |R_i|$, $b_i := |R_i^*|$, $\tau_i := B_{i-1}$, $i = 1, \ldots, p$ yields in (1)

$$\text{OL}(\sigma) \leq 2 \cdot 3^{1/g} \sum_{i=1}^{p} B_{i-1} |R_i| \leq 2 \cdot 3^{1/g} \sum_{i=1}^{p} B_{i-1} |R_i^*|. \tag{2}$$

Let $C_j^*$ be the optimal off-line completion time of request $\sigma_j$. For each $\sigma_j$ denote by $(B_{\phi_j}, B_{\phi_{j+1}}]$ the interval that contains $C_j^*$. This inserted in (2) yields

$$\text{OL}(\sigma) \leq 2 \cdot 3^{1/g} \sum_{j=1}^{m} B_{\phi_j} \leq 2 \cdot 3^{1/g} \sum_{j=1}^{m} C_j^* = 2 \cdot 3^{1/g} \cdot \text{OPT}(\sigma).$$

□

**Corollary 3.2.** *There is a deterministic* 6*-competitive online algorithm for the* $k$*-TRP.*

We can improve the bounds slightly such as to match the $(1+\sqrt{2})^2$-competitive algorithm [16] for the TRP but at the expense of increased technical details.

### 3.3 Lower Bounds

**Theorem 3.3.** *Any randomized* $c$*-competitive online algorithm for the* $(k, k^*)$*-TSP or the* $(k, k^*)$*-TRP has* $c \geq 2$*.*

*Proof.* Consider the metric space induced by a star graph with $m$ unit-length rays, the origin being the center of the star. No request is given until time 1. At time 1, the adversary gives a request on an edge chosen uniformly at random, at distance 1 from the origin. The expected makespan for the adversary is 1. For the online algorithm, we say that a server *guards* a ray if at time 1 the server is located on the ray, but not at the center of the star. Then the makespan is at least 2 if no server guards the ray where the request is released, and at least 1 otherwise. But $k$ servers can guard at most $k$ rays, so

$$\mathbb{E}[\text{OL}(\sigma)] \geq 2 \cdot \left(1 - \frac{k}{m}\right) + 1 \cdot \frac{k}{m} \geq 2 - \frac{k}{m}$$

and the result follows by Yao's principle, since $m$ can be arbitrarily large. □

Notice that this lower bound is independent of the values $k$ and $k^*$. A consequence of this is that the upper bounds of Sections 3.1 and 3.2 are essentially best possible when $k \gg k^*$, as in that case they both approach 2.

## 4 Algorithms for the Real Line

### 4.1 An Asymptotically Optimal Algorithm

**Theorem 4.1.** *There is a deterministic online algorithm with competitive ratio* $1 + O((\log k)/k)$ *for both the* $k$*-TSP and the* $k$*-TRP on the real line.*

As a preliminary, we prove a similar result on the *halfline*. Let $g_k$ be the unique root greater than 1 of the equation $z^k(z-1) = 3z - 1$.

**Lemma 4.1.** *GPS (Algorithm 3) is* $g_k$*-competitive for* $k$*-TSP and* $k$*-TRP on the halfline.*

**Algorithm 3.** Geometric Progression Speeds (GPS)

As a preprocessing step, the algorithm delays every request $(r, x)$ for which $x \geq r$ to time $x$; that is, the release date of each request $(r, x)$ is reset at $r' := \max\{r, x\}$ (the *modified release date*).
Then, let $g_k$ be the unique root greater than 1 of the equation $g_k^k = \frac{3g_k - 1}{g_k - 1}$ and define $\alpha_j = g_k^{j-k-1}$ for $j \in \{2, 3, \ldots, k\}$. For every $j > 1$, server $s_j$ departs at time 0 from $o$ at speed $\alpha_j$ and never turns back. The first server $s_1$ waits in $o$ until the first request $(r_0, x_0)$ is released with $0 < x_0 < s_2(r'_0)$. For $i \geq 0$, define $t_i = g_k^i r'_0$. During any interval $[t_{i-1}, t_i]$, $s_1$ moves at full speed first from $o$ to $\frac{g_k - 1}{2} t_{i-1}$ and then back to $o$.

*Proof.* First, notice that the modified release date of a request is a lower bound on its completion time. Thus it is enough to prove that, for every request $(r, x)$, the time at which it is served is at most $g_k r'$.

For $1 < j < k$, we say that a request $(r, x)$ is in *zone* $j$ if $\alpha_j \leq x/r' < \alpha_{j+1}$. We also say that a request is in zone 1 if $x/r' < \alpha_2$, and that it is in zone $k$ if $x/r' \geq \alpha_k$. By construction, every request is in some zone and a request in zone $j$ will be eventually served by server $s_j$.

For a request $(r, x)$ in a zone $j$ with $1 < j < k$, since the request is served by server $s_j$ at time $x/\alpha_j$ and since $x \leq \alpha_{j+1} r$, the ratio between completion time and modified release date is at most $\alpha_{j+1}/\alpha_j = g_k$. Similarly, for a request in zone $k$, since $x \leq r'$, the ratio between completion time and modified release date is at most $1/\alpha_k = g_k$.

It remains to give a bound for requests in zone 1. Take any such request, i.e., a request $(r, x)$ such that $x < \alpha_2 r'$ and suppose it is served at time $\tau \in [t_{i-1}, t_i]$ for some $i$. If $r' \geq t_{i-1}$, then, since $\tau \leq t_i$, the ratio between $\tau$ and $r'$ is at most $g_k$ by definition of $t_i$, $i \geq 0$.

If $r' < t_{i-1}$, then, since $\tau > t_{i-1}$, only two possible cases remain. First, the situation that $x > \frac{g_k - 1}{2} t_{i-2}$. Since $\tau = t_{i-1} + x$ and $r' \geq x/\alpha_2$, we have

$$\frac{\tau}{r'} \leq \frac{x + t_{i-1}}{x/\alpha_2} \leq \alpha_2 \left(1 + \frac{2 g_k t_{i-2}}{(g_k - 1) t_{i-2}}\right) = \alpha_2 \frac{3g_k - 1}{g_k - 1} = \alpha_2 g_k^k = g_k.$$

In the second situation, $x \leq \frac{g_k - 1}{2} t_{i-2}$. Then $r'$ must be such that $s_1$ was already on its way back to 0 during $[t_{i-2}, t_{i-1}]$, in particular $r' \geq g_k t_{i-2} - x$. Thus,

$$\tau / r' \leq \frac{g_k t_{i-2} + x}{g_k t_{i-2} - x} \leq \frac{3g_k - 1}{g_k + 1} \leq g_k. \qquad \square$$

The algorithm for the real line simply splits the $k$ servers evenly between the two halflines, and uses GPS on each halfline.

**Lemma 4.2.** *For any $k \geq 2$, SGPS (Algorithm 4) is $g_{\lfloor k/2 \rfloor}$-competitive for the $k$-TSP and the $k$-TRP on the line.*

*Proof.* The only lower bounds on the offline cost that we used in the proof of Lemma 4.1 were the distance of every request from $o$ and the release date of

**Algorithm 4.** Split Geometric Progression Speeds (SGPS)

Arbitrarily assign $\lceil k/2 \rceil$ servers to $\mathbb{R}_+$ and $\lfloor k/2 \rfloor$ servers to $\mathbb{R}_-$. On each of the two halflines, apply Algorithm 3 independently (i.e., ignoring the requests and the servers in the other halfline).

every request. They are valid independent of the number of offline servers. In particular, they hold if the number of offline servers is twice the number of online servers. Thus, we can analyze the competitiveness of the online servers on each of the two halflines separately and take the worst of the two competitive ratios. □

**Lemma 4.3.** *For any $k \geq 1$, $g_k \leq 1 + \frac{2\log k+3}{k}$.*

*Proof.* We defined $g_k$ as the unique root greater than 1 of $z^k = 1 + \frac{2z}{z-1}$. Since $\lim_{z\to\infty} z^k > \lim_{z\to\infty} 1 + \frac{2z}{z-1}$, it suffices to prove that $z_0 := 1 + \frac{2\log k+3}{k}$ satisfies $z_0^k \geq 1 + \frac{2z_0}{z_0-1}$. The binomial theorem and the standard fact that $\binom{k}{j} \geq \frac{k^j}{j^j}$ yield

$$z_0^k - 1 = \sum_{j=1}^{k} \binom{k}{j} \frac{(2\log k+3)^j}{k^j} \geq \sum_{j=1}^{k} \frac{(2\log k+3)^j}{j^j} \geq \sum_{j=1}^{\lfloor \log k \rfloor + 1} \left(\frac{2\log k+3}{j}\right)^j$$
$$\geq \sum_{j=1}^{\lfloor \log k \rfloor + 1} 2^j \geq 2^{\log k+1} - 2 = 2k - 2.$$

Now it can be verified that for all $k > 2$, $2k - 2 > \frac{2k}{2\log k+3} + 2 = \frac{2z_0}{z_0-1}$. Finally, the bound also holds for $k \in \{1, 2\}$ as seen by explicitly finding $g_1$ and $g_2$. □

Theorem 4.1 now follows from Lemma 4.2 and Lemma 4.3.

### 4.2 Lower Bounds

**Theorem 4.2.** *Any randomized $c$-competitive online algorithm for the $k$-TSP or the $k$-TRP on the line has $c \geq 1 + 1/2k = 1 + \Omega(1/k)$.*

*Proof.* The adversary gives a single request at time 1, in a point drawn uniformly at random from the interval $[-1, 1]$. The expected optimal cost is obviously 1. Thus, by Yao's principle it suffices to show that $\mathbb{E}[\text{OL}(\sigma)] \geq 1 + 1/2k$.

In order to bound $\mathbb{E}[\text{OL}(\sigma)]$, let $f(x) = \min_{j\in\{1,\ldots,k\}} d(x, s_j(1))$. Notice that $1 + f(x)$ is a lower bound on the cost paid by the online algorithm, assuming that the request was given at $x$. In terms of expected values,

$$\mathbb{E}[\text{OL}(\sigma)] \geq \mathbb{E}[1 + f(x)] = 1 + \frac{1}{2}\int_{-1}^{1} f(x)dx.$$

Thus, we want to find the minimum value of the area below $f$ in $[-1, 1]$. That area is minimized when the servers are evenly spread inside the interval and at distance $1/k$ from the extremes, in which case its value is $1/k$. □

## 5 Lower Bounds on the Plane

Comparing the results in Section 3 with those in Section 4, we see that while in general spaces the competitive ratio of both the $k$-TSP and the $k$-TRP always remains lower bounded by 2, on the real line we can achieve $1 + o(1)$ asymptotically. A natural question is whether on a low-dimensional space like the Euclidean plane we can also achieve $1 + o(1)$ competitiveness. In this section we answer this question negatively.

**Theorem 5.1.** *Any randomized $c$-competitive online algorithm for the $k$-TSP on the plane has $c \geq 4/3$.*

*Proof.* As a crucial ingredient of the proof we introduce a new kind of request, which is located in a single point $x$ of the space but has an arbitrarily long processing time $p$ (this processing time can be divided among the servers processing the request). We show how this can be emulated in the Euclidean plane with arbitrarily good precision by giving a high enough number of requests packed inside an arbitrarily small square around $x$.

Fix some arbitrary $\epsilon > 0$. Consider a square with sidelength $s = \sqrt{\epsilon p}$ centered around $x$. The square can be partitioned in $s^2/\epsilon^2$ smaller squares of sidelength $\epsilon$. In the center of each of these smaller squares we give a request. Notice that the distance between any pair of such requests is at least $\epsilon$. Thus, the sum of the times required for any $k$ servers to serve all requests is at least $(\frac{s^2}{\epsilon^2} - k)\epsilon$, no matter where the servers start (the $-k\epsilon$ term reflects the possible saving each server could have by starting arbitrarily close to the *first* request he serves).

For $\epsilon$ tending to zero, the requests converge to the point $x$ and the total processing time needed converges to $p$. If the starting points of the servers are most favourable, an algorithm could finish serving all requests in time $p/k$.

We show how to use such a "long" request to achieve our lower bound. At time 1, the adversary gives a long request of processing time $p = 2k$ in a point drawn uniformly at random from $\{(1, 0), (-1, 0)\}$. The expected optimal cost is $1 + p/k = 3$. By Yao's principle, it remains to prove that $\mathbb{E}[\text{OL}(\sigma)] \geq 4$.

Since there is a single long request, we can assume wlog that all the online servers will move to the request and contribute to serving it. Since $p = 2k$, the server that will contribute most to the service will have to spend time at least $2k/k = 2$ in $x$, and this is enough for any other server to arrive and give a contribution (since at time 1 no server can be farther than 2 from $x$).

Suppose wlog that the servers are numbered in order of nondecreasing distance to $x$ and let $d_i = d(x, s_i(1))$. $\text{OL}(\sigma) \geq 1 + t_0$, with $t_0$ the time needed for the servers to completely serve the request, i.e., the time when its remaining processing time is zero. Thus, $t_0$ satisfies $\sum_{i=1}^{k-1} i(d_{i+1} - d_i) + k(t_0 - d_k) = p$ , since during interval $[d_i, d_{i+1})$ exactly $i$ servers are processing the request. Hence,

$$kt_0 = p + kd_k - \sum_{i=1}^{k-1} i(d_{i+1} - d_i) = p + \sum_{i=1}^{k} d_i.$$

Now consider the positions of the online servers at time 1 inside the ball of radius 1 around the origin. Regarding points as vectors in $\mathbb{R}^2$, $d_i$ can be written as $||s_i(1) - x||$ (here $||\cdot||$ denotes the $L_2$ norm). Then

$$\sum_{i=1}^{k} d_i = \sum_i ||s_i(1) - x|| \geq ||\sum_i (s_i(1) - x)||$$
$$= k||\frac{1}{k}\sum_i s_i(1) - x|| = k||b - x|| = k \cdot d(b, x),$$

where $b = \frac{1}{k}\sum_i s_i(1)$ is the centroid of the $s_i(1)$. Hence,

$$\mathbb{E}[\text{OL}(\sigma)] \geq 1 + \mathbb{E}[t_0] \geq 1 + p/k + \mathbb{E}[d(b, x)] =$$
$$= 3 + (1/2)\, d(b, (1, 0)) + (1/2)\, d(b, (-1, 0))$$
$$\geq 3 + (1/2)\, d((1, 0), (-1, 0)) = 4. \qquad \square$$

A similar technique gives an analogous lower bound for the $k$-TRP on the plane.

**Theorem 5.2.** *Any randomized $c$-competitive online algorithm for the $k$-TRP on the plane has $c \geq 5/4$.*

## 6 Conclusions and Open Problems

After analyzing the differences between multiple and single server variants, we can conclude that sometimes having multiple servers is more beneficial to the online algorithm than to the offline adversary. In some cases, including the traveling repairman problem on the line, the online algorithms can approach the offline cost when there are enough servers. In more general spaces, these extremely favorable situation cannot occur. Still in some intermediate cases, like the Euclidean plane, it is conceivable that the competitive ratios become lower than those of the corresponding single server problems. We leave the analysis of the competitive ratio in these situations as an open problem.

## References

[1] N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In H. Reichel and S. Tison, editors, *Proc. 17th Symp. on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 639–650. Springer-Verlag, 2000.

[2] G. Ausiello, V. Bonifaci, and L. Laura. The on-line asymmetric traveling salesman problem. In F. Dehne, A. López-Ortiz, and J. Sack, editors, *Proc. 9th Workshop on Algorithms and Data Structures*, volume 3608 of *Lecture Notes in Computer Science*, pages 306–317. Springer-Verlag, 2005.

[3] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 2001.

[4] M. Blom, S. O. Krumke, W. E. de Paepe, and L. Stougie. The online TSP against fair adversaries. *INFORMS Journal on Computing*, 13(2):138–148, 2001.

[5] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
[6] A. Borodin and R. El-Yaniv. On randomization in online computation. *Information and Computation*, 150:244–267, 1999.
[7] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proc. 44th Symp. on Foundations of Computer Science*, pages 36–45, 2003.
[8] B. Chen and A. P. A. Vestjens. Scheduling on identical machines: How good is LPT in an on-line setting? *Operations Research Letters*, 21:165–169, 1998.
[9] J. R. Correa and M. R. Wagner. LP-based online scheduling: From single to parallel machines. In *Integer programming and combinatorial optimization*, volume 3509 of *Lecture Notes in Computer Science*, pages 196–209. Springer-Verlag, 2005.
[10] J. Fakcharoenphol, C. Harrelson, and S. Rao. The $k$-traveling repairman problem. In *Proc. 14th Symp. on Discrete Algorithms*, pages 655–664, 2003.
[11] E. Feuerstein and L. Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.
[12] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.
[13] D. Hauptmeier, S. O. Krumke, and J. Rambau. The online dial-a-ride problem under reasonable load. In G. Bongiovanni, G. Gambosi, and R. Petreschi, editors, *Proc. 4th Italian Conference on Algorithms and Complexity*, volume 1767 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, 2000.
[14] R. Jothi and B. Raghavachari. Minimum latency tours and the $k$-traveling repairmen problem. In M. Farach-Colton, editor, *Proc. 6th Symp. Latin American Theoretical Informatics*, volume 2976 of *Lecture Notes in Computer Science*, pages 423–433. Springer-Verlag, 2004.
[15] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47:214–221, 2000.
[16] S. O. Krumke, W. E. de Paepe, D. Poensgen, and L. Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295(1-3):279–294, 2003.
[17] S. O. Krumke, L. Laura, M. Lipmann, A. Marchetti-Spaccamela, W. E. de Paepe, D. Poensgen, and L. Stougie. Non-abusiveness helps: an $O(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In K. Jansen, S. Leonardi, and V. V. Vazirani, editors, *Proc. 5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization*, volume 2462 of *Lecture Notes in Computer Science*, pages 200–214. Springer-Verlag, 2002.
[18] E. L. Lawler, J. K. Lenstra, A. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, England, 1985.
[19] M. Lipmann. *On-Line Routing*. PhD thesis, Technical Univ. Eindhoven, 2003.
[20] M. Lipmann, X. Lu, W. E. de Paepe, R. A. Sitters, and L. Stougie. On-line dial-a-ride problems under a restricted information model. *Algorithmica*, 40:319–329, 2004.
[21] M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.
[22] J. Sgall. On-line scheduling. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, pages 196–231. Springer, 1998.
[23] L. Stougie and A. P. A. Vestjens. Randomized on-line scheduling: How low can't you go? *Operations Research Letters*, 30:89–96, 2002.

# Theoretical Evidence for the Superiority of LRU-2 over LRU for the Paging Problem*

Joan Boyar, Martin R. Ehmsen, and Kim S. Larsen

Department of Mathematics and Computer Science
University of Southern Denmark, Odense, Denmark
{joan,ehmsen,kslarsen}@imada.sdu.dk

**Abstract.** The paging algorithm LRU-2 was proposed for use in database disk buffering and shown experimentally to perform better than LRU [O'Neil, O'Neil, and Weikum, 1993]. We compare LRU-2 and LRU theoretically, using both the standard competitive analysis and the newer relative worst order analysis. The competitive ratio for LRU-2 is shown to be $2k$ for cache size $k$, which is worse than LRU's competitive ratio of $k$. However, using relative worst order analysis, we show that LRU-2 and LRU are asymptotically comparable in LRU-2's favor, giving a theoretical justification for the experimental results.

## 1 Introduction

On many layers in a computer system, one is faced with maintaining a subset of memory units from a relatively slow memory in a significantly smaller fast memory. For ease of terminology, we refer to the fast memory as the *cache* and to the memory units as *pages*. The cache will have size $k$, meaning that it can hold at most $k$ pages at one time. Pages are requested by the user (possibly indirectly by an operating or a database system) and the requests must be treated one at a time without knowledge of future requests. This makes the problem an *on-line* problem [2]. If a requested page is already in cache, this is referred to as a *hit*. Otherwise, we have a *page fault*. The treatment of a request must entail that the requested page reside in cache. Thus, the only freedom is the choice of a page to evict from cache in order to make room for the requested page in the case of a page fault. An algorithm for this problem is referred to as a *paging algorithm*. Other names for this in the literature are "eviction strategy" or "replacement policy". Various cost models for this problem have been studied. We focus on the classic model of minimizing the number of page faults. The problem is of great importance in database systems where it is often referred to as the *database disk buffering problem*. See [2] for an overview of the paging problem, cost models, and paging algorithms in general.

Probably the most well-known paging algorithm is LRU (Least-Recently-Used), which on a page fault evicts the least recently used page from cache.

* This work was supported in part by the Danish Natural Science Research Council (SNF).

T. Erlebach and C. Kaklamanis (Eds.): WAOA 2006, LNCS 4368, pp. 95–107, 2006.

The experience from real-life request sequences is that overall LRU performs better than all other paging algorithms which have been proposed up until the introduction of LRU-2 [12]. On a page fault, LRU-2 evicts the page with the least recent second to last request (if there are pages in cache which have been requested only once, the least recently used of these is evicted). Compelling empirical evidence is given in [12] in support of the superiority of LRU-2 over LRU in database systems. We return to this issue below. Since the introduction of LRU-2, there have been other proposals for better paging algorithms; for example [8].

In the on-line community, there are, to our knowledge, no published results on LRU-2. We assume that this is because it has not been possible to explain the experimental results theoretically. In this paper, we provide a theoretical justification of LRU-2's superiority over LRU. More specifically, we show using relative worst order analysis [4] that LRU-2 and LRU are asymptotically comparable in LRU-2's favor. In establishing this result, we prove a general result giving on upper bound on how well any algorithm can perform relative to LRU.

It is well-known that analysis of the paging problem is particularly problematic for the most standard quality measure for on-line algorithms, the competitive ratio [9,13]. This has lead researchers to investigate alternative methods. See a long list of these in [1]. However, these methods are mostly only applicable to the paging problem.

In contrast, it has been demonstrated that the relative worst order ratio is generally applicable. In most cases, the relative worst order ratio makes the same distinction between algorithms as the competitive ratio does. However, the following is a list of results, where the relative worst order ratio has distinguished algorithms in a situation where the competitive ratio cannot distinguish or even in some cases favors the "wrong" algorithm. This is not an exclusive list; we merely highlight one result from each of these on-line problems:

- For classical bin packing, Worst-Fit is better than Next-Fit [4].
- For dual bin packing, First-Fit is better than Worst-Fit [4].
- For paging, LRU is better than FWF (Flush-When-Full) [5].
- For scheduling, minimizing makespan on two related machines, a post-greedy algorithm is better than scheduling all jobs on the fast machine [7].
- For bin coloring [11], a natural greedy-type algorithm is better than just using one open bin at a time [10].
- For proportional price seat reservation, First-Fit is better than Worst-Fit [6].

We refer the reader to the referenced papers for details and more results. Here, we merely want to point out that the relative worst order ratio is an appropriate tool to apply to on-line problems in general and the paging problem in particular.

LRU-2, along with previous results and testing of the algorithm, is described in Section 2. Its competitive ratio is proven to be $2k$ in Section 3, showing that LRU-2 has a suboptimal competitive ratio, in comparison to LRU's competitive ratio of $k$. However, in Section 4, relative worst order analysis is applied showing that LRU-2 is asymptotically comparable to LRU in LRU-2's favor, providing

the theoretical justification for LRU-2's superiority. A result which may be of independent interest bounds $c_{\mathrm{LRU},\mathbb{A}}$, the factor by which any algorithm $\mathbb{A}$ can be better than LRU using relative worst order analysis: $c_{\mathrm{LRU},\mathbb{A}} \leq \frac{k+1}{2}$.

## 2 LRU-2 and Experimental Results

In [12], a new family of paging algorithms, LRU-K, is defined. Here, K is a constant which defines the algorithm. On a page fault, LRU-K evicts the page with the least recent K'th last request (for $K = 1$, LRU-K is LRU). If there are pages in cache which have been requested fewer than K times, then some subsidiary policy must be employed for those pages. In [12], LRU is suggested as a possible subsidiary policy. However, it would also be natural to recursively use LRU-(K − 1). For the case of K = 2, this is the same.

The authors' motivation for considering LRU-2 (or LRU-K in general for various K) is that LRU does not discriminate between pages with very frequent versus very infrequent references. Both types will be held in cache for a long time once they are brought in. This can be at the expense of pages with very frequent references.

The algorithm LFU (Least-Frequently-Used) which evicts the page which is least frequently used is the ultimate algorithm in the direction of focusing on frequency, but this algorithm appears to adjust too slowly to changing patterns in the request sequence [13]. The family of algorithms, LRU = LRU-1, LRU-2, LRU-3, ... with recursive subsidiary policies can be viewed as approaching the behavior of LFU.

A conscientious testing in [12] of particularly LRU-2 and LRU-3 up against LRU and LFU lead the authors to conclude that LRU-2 is the algorithm of choice.

The algorithms are tested in a real database system environment using random references from a Zipfian distribution, using real-life data from a CODASYL database system, and finally using data generated to simulate request sequences which would arise from selected applications where LRU-2 is expected to improve performance.

LRU-2 and LRU-3 perform very similarly and in all cases significantly better than the other algorithms. Many test results are reported which can be viewed in different ways. If one should summarize the results in one sentence, we would say that LRU and LFU need 50–100% extra cache space in order to approach the performance of LRU-2.

## 3 Competitive Ratio Characterizations

Let $\mathbb{A}(I)$ denote the number of page faults $\mathbb{A}$ has on request sequence $I$. The standard measure for the quality of on-line algorithms is the *competitive ratio*: $\mathrm{CR}(\mathbb{A})$ of $\mathbb{A}$ is $\mathrm{CR}(\mathbb{A}) = \inf\{c \mid \exists b\colon \forall I\colon \mathbb{A}(I) \leq c \cdot \mathrm{OPT}(I) + b\}$, where OPT denotes an optimal off-line algorithm [9,13].

LRU is known to be both a conservative algorithm [14] and a marking algorithm [3]. Both types of algorithms have competitive ratio $k$. The following request sequence, $\langle (p_1, p_1), (p_2, p_2), ..., (p_{k+1}, p_{k+1}), (p_1, p_2, p_1, p_2) \rangle$, shows that LRU-2 belongs to neither of these classes, since it faults on all of the last four requests. Thus, it is not obvious that its competitive ratio is $k$. In fact the lemma below shows that it is larger than $k$.

**Lemma 1.** *The competitive ratio of* LRU-2 *is at least* $2k$ *for* $k$ *even.*

*Proof.* Assume that there are $k+1$ distinct pages, $p_1, p_2, ..., p_{k+1}$, in slow memory, and that $k$ is even.

Let

$$P_1 = \langle (p_2, p_2), (p_3, p_3), \ldots, (p_{k+1}, p_{k+1}) \rangle$$
$$P_2 = \langle (p_2, p_2), (p_3, p_3), \ldots, (p_k, p_k), (p_1, p_1) \rangle$$

and define the request sequence $I_l$ by

$$\langle P_1, (p_1, p_2, p_1, p_2), (p_3, p_4, p_3, p_4), \ldots, (p_{k-1}, p_k, p_{k-1}, p_k),$$
$$P_2, (p_{k+1}, p_2, p_{k+1}, p_2), (p_3, p_4, p_3, p_4), \ldots, (p_{k-1}, p_k, p_{k-1}, p_k) \rangle^l.$$

After LRU-2 processes $P_1$, the page $p_1$ will not be in cache. Considering any block $(p_i, p_{i+1}, p_i, p_{i+1})$, for $1 \leq i \leq k+1$ in $I_l$ (where $(k+1)+1$ will be considered 2), it follows inductively that the page $p_i$ is not in LRU-2's cache on the first request in that block. The faults on $p_i$ cause $p_{i+1}$ to be evicted and vice versa until the second fault on $p_{i+1}$, which causes $p_{i+2}$ (or $p_3$, if $i = k+1$) to be evicted. Thus, LRU-2 faults $k$ times during the first occurrence of $P_1$, never on $P_1$ or $P_2$ after that, and on all $4kl$ of the remaining requests. OPT, on the other hand, faults $k$ times during the first occurrence of $P_1$. It also faults on requests to $p_1$ immediately following $P_1$ and evicts $p_{k+1}$ each time. Similarly, on the requests to $p_{k+1}$ immediately following $P_2$, it evicts $p_1$. Thus it faults $k + 2l$ times in all. Since $l$ can be arbitrarily large, this gives a ratio of $2k$ asymptotically. □

**Lemma 2.** LRU-2 *is* $2k$*-competitive.*

*Proof.* First notice that it is enough to prove that in each $k$-phase (a maximal subsequence of consecutive requests containing exactly $k$ distinct pages) of any sequence $I$, LRU-2 faults at most two times on each of the $k$ different pages requested in that phase. Suppose, for the sake of contradiction, that LRU-2 faults more than two times on some page in a phase $P$. Let $p$ be the first page in $P$ with more than two faults. At some point between the second and third faults on $p$, $p$ must have been evicted by a request to some page $q$. The page $q$ is one of the $k$ pages in $P$. Thus, at this point there must be some page $r$ in cache which is not in $P$. The second to last request to $r$ must be before the start of $P$ and thus before the second to last request to $p$. Hence, $p$ could not have been evicted at this point. This gives a contradiction, so there are at most $2k$ faults in any $k$-phase. □

The following theorem follows immediately from the previous two results:

**Theorem 1.** CR(LRU-2) $= 2k$.

## 4 Relative Worst Order Characterizations

Now we show the theoretical justification for the empirical result that LRU-2 performs better than LRU. In order to do this, we use a different measure for the quality of on-line algorithms, the relative worst order ratio [4,5], which has previously [4,5,7,10,6] proven capable of differentiating between algorithms in other cases where the competitive ratio failed to give the "correct" result. Instead of comparing on-line algorithms to an optimal off-line algorithm (and then comparing their competitive ratios), two on-line algorithms are compared directly. However, instead of comparing their performance on the exact same sequence, they are compared on their respective worst permutations of the same sequence:

**Definition 1.** *Let $\sigma(I)$ denote a permutation of the sequence $I$, let $\mathbb{A}$ and $\mathbb{B}$ be algorithms for the paging problem, and let $\mathbb{A}_W(I) = \max_\sigma\{\mathbb{A}(\sigma(I))\}$. Let $S_1$ and $S_2$ be statements about algorithms $\mathbb{A}$ and $\mathbb{B}$ defined in the following way.*

$$S_1(c) \triangleq \exists b\colon \forall I\colon \mathbb{A}_W(I) \leq c\,\mathbb{B}_W(I) + b$$
$$S_2(c) \triangleq \exists b\colon \forall I\colon \mathbb{A}_W(I) \geq c\,\mathbb{B}_W(I) - b$$

*The* relative worst order ratio $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ *of algorithm $\mathbb{A}$ to algorithm $\mathbb{B}$ is defined if $S_1(1)$ or $S_2(1)$ holds.*

$$\textit{If } S_1(1) \textit{ holds, then } \mathrm{WR}_{\mathbb{A},\mathbb{B}} = \sup\{r \mid S_2(r)\}\,, \textit{ and}$$
$$\textit{if } S_2(1) \textit{ holds, then } \mathrm{WR}_{\mathbb{A},\mathbb{B}} = \inf\{r \mid S_1(r)\}\,.$$

The statements $S_1(1)$ and $S_2(1)$ check that the one algorithm is always at least as good as the other on every sequence (on their respective worst permutations). When one of them holds, the relative worst order ratio is a bound on how much better the one algorithm can be. In some cases, however, the first algorithm can do significantly better better than the second, while the second can sometimes do marginally better than the first. In such cases, we use the following definitions (from [5], but restricted to the paging problem here) and show that the two algorithms are asymptotically comparable in favor of the first algorithm.

**Definition 2.** *Let $\mathbb{A}$ and $\mathbb{B}$ be algorithms for the paging problem, and let the statement $S_1(c)$ be defined as above. If there exists a positive constant $c$ such that $S_1(c)$ is true, let $c_{\mathbb{A},\mathbb{B}} = \inf\{r \mid S_1(r)\}$. Otherwise, $c_{\mathbb{A},\mathbb{B}}$ is undefined.*

- *If $c_{\mathbb{A},\mathbb{B}}$ and $c_{\mathbb{B},\mathbb{A}}$ are both defined, $\mathbb{A}$ and $\mathbb{B}$ are* $(c_{\mathbb{A},\mathbb{B}}, c_{\mathbb{B},\mathbb{A}})$-related.
- *If $c_{\mathbb{A},\mathbb{B}}$ is defined and $c_{\mathbb{B},\mathbb{A}}$ is undefined, $\mathbb{A}$ and $\mathbb{B}$ are* $(c_{\mathbb{A},\mathbb{B}}, \infty)$-related.
- *If $c_{\mathbb{A},\mathbb{B}}$ is undefined and $c_{\mathbb{B},\mathbb{A}}$ is defined, $\mathbb{A}$ and $\mathbb{B}$ are* $(\infty, c_{\mathbb{B},\mathbb{A}})$-related.

*$\mathbb{A}$ and $\mathbb{B}$ are* asymptotically comparable*, if*

$$\left(\lim_{k\to\infty}\{c_{\mathbb{A},\mathbb{B}}\} \leq 1 \wedge \lim_{k\to\infty}\{c_{\mathbb{B},\mathbb{A}}\} \geq 1\right) \vee \left(\lim_{k\to\infty}\{c_{\mathbb{A},\mathbb{B}}\} \geq 1 \wedge \lim_{k\to\infty}\{c_{\mathbb{B},\mathbb{A}}\} \leq 1\right)$$

*where $k$ is the size of the cache.*

*If $\mathbb{A}$ and $\mathbb{B}$ are asymptotically comparable algorithms, then $\mathbb{A}$ and $\mathbb{B}$ are* asymptotically comparable in $\mathbb{A}$'s favor *if* $\lim_{k\to\infty}\{c_{\mathbb{B},\mathbb{A}}\} > 1$.

The relation, being asymptotically comparable in the first algorithm's favor, is transitive, so it gives a well defined means of comparing on-line algorithms.

**Lemma 3.** *Asymptotically comparable in an on-line algorithm's favor is a transitive relation.*

*Proof.* Assume that three algorithms $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$ are related such that $\mathbb{A}$ is asymptotically comparable to $\mathbb{B}$ in $\mathbb{A}$'s favor and $\mathbb{B}$ is asymptotically comparable to $\mathbb{C}$ in $\mathbb{B}$'s favor.

We need to show that $\mathbb{A}$ is asymptotically comparable to $\mathbb{C}$ in $\mathbb{A}$'s favor, i.e.,

$$(\lim_{k\to\infty} c_{\mathbb{A},\mathbb{C}} \leq 1) \wedge (\lim_{k\to\infty} c_{\mathbb{C},\mathbb{A}} > 1).$$

Since $\mathbb{A}$ is asymptotically comparable to $\mathbb{B}$ in $\mathbb{A}$'s favor and $\mathbb{B}$ is asymptotically comparable to $\mathbb{C}$ in $\mathbb{B}$'s favor, we know that

$$(\lim_{k\to\infty} c_{\mathbb{A},\mathbb{B}} \leq 1) \wedge (\lim_{k\to\infty} c_{\mathbb{B},\mathbb{A}} > 1)$$

and

$$(\lim_{k\to\infty} c_{\mathbb{B},\mathbb{C}} \leq 1) \wedge (\lim_{k\to\infty} c_{\mathbb{C},\mathbb{B}} > 1).$$

It follows that

$$\begin{aligned}
1 &\geq (\lim_{k\to\infty} c_{\mathbb{A},\mathbb{B}})(\lim_{k\to\infty} c_{\mathbb{B},\mathbb{C}}) \\
&= \lim_{k\to\infty} (\inf\{c_1 : \exists b_1 \forall I : \mathbb{A}_W(I) \leq c_1 \mathbb{B}_W(I) + b_1\} \cdot \\
&\qquad \inf\{c_2 : \exists b_2 \forall I : \mathbb{B}_W(I) \leq c_2 \mathbb{C}_W(I) + b_2\}) \\
&= \lim_{k\to\infty} \inf\{c_1 c_2 : \exists b_1, b_2 \forall I : \mathbb{A}_W(I) \leq c_1 \mathbb{B}_W(I) + b_1 \wedge \\
&\qquad\qquad \mathbb{B}_W(I) \leq c_2 \mathbb{C}_W(I) + b_2\} \\
&= \lim_{k\to\infty} \inf\{c : \exists b \forall I : \mathbb{A}_W(I) \leq c \mathbb{C}_W(I) + b\} \\
&= \lim_{k\to\infty} c_{\mathbb{A},\mathbb{C}}.
\end{aligned}$$

In the above, we use the fact that the $c_1$'s and $c_2$'s can be assumed to be non-negative.

A similar argument shows that $1 < \lim_{k\to\infty} c_{\mathbb{C},\mathbb{A}}$. □

We proceed to show that LRU-2 and LRU are asymptotically comparable in LRU-2's favor. First, we show that LRU-2 can perform significantly better than LRU on some sets of input.

**Theorem 2.** *There exists a family of sequences $I_n$ of page requests and a constant $b$ such that*

$$\text{LRU}_W(I_n) \geq \frac{k+1}{2}\text{LRU-2}_W(I_n) - b,$$

*and* $\lim_{n\to\infty} \text{LRU}_W(I_n) = \infty$.

*Proof.* Let $I_n$ consist of $n$ phases, where in each phase, the first $k-1$ requests are to the $k-1$ pages $p_1, p_2, \ldots, p_{k-1}$, always in that order, and the last two requests are to completely new pages. LRU will fault on every page, so it will fault $n(k+1)$ times.

Regardless of the order this sequence is given in, LRU-2 will never evict any page $p' \in \{p_1, p_2, \ldots, p_{k-1}\}$ after the second request to $p'$. This follows from the fact that there are at most $k-1$ pages in cache with two or more requests at any point in time. Hence, when LRU-2 faults there is at least one page in cache with only one request in its history. By definition, LRU-2 must use its subsidiary policy when there exists pages in cache with less than two requests, and it must choose among these pages. This means that LRU-2 faults at most $2(k-1)+2n$ times.

Asymptotically, the ratio is $\frac{k+1}{2}$. □

The above ratio of $\frac{k+1}{2}$ cannot be improved. In fact no paging algorithm $\mathbb{A}$ can be $(c_{\mathbb{A},\mathrm{LRU}}, c_{\mathrm{LRU},\mathbb{A}})$-related to LRU with $c_{\mathrm{LRU},\mathbb{A}} > \frac{k+1}{2}$.

**Theorem 3.** *For any paging algorithm* $\mathbb{A}$*,*

$$c_{\mathrm{LRU},\mathbb{A}} \leq \frac{k+1}{2}.$$

*Proof.* Suppose there exists a sequence $I$, where LRU faults $s$ times on its worst permutation, $I_{\mathrm{LRU}}$, $\mathbb{A}$ faults $s'$ times on its worst permutation, $I_{\mathbb{A}}$, and $s > \frac{k+1}{2}s'$. As proven in [5], there exists a worst permutation $I_f$ of $I_{\mathrm{LRU}}$ with respect to LRU where all faults appear before all hits. Let $I_1$ be the prefix of $I_f$ consisting of the $s$ faults. Partition the sequence $I_1$ into subsequences of length $k+1$ (except possibly the last which may be shorter). We process these subsequences one at time, possibly reordering some of them, so that $\mathbb{A}$ faults at least twice on all, except possibly the last. (Note that since $\mathbb{A}$ will fault on the first $k+1$ requests, if $k \geq 3$, this last incomplete subsequence can be ignored. Otherwise, it contributes at most an additive constant of $k$ to the inequality in statement $S_1(\frac{k+1}{2})$.) The first subsequence need not be reordered. Suppose the first $i$ subsequences have been considered and consider the $i+1$st, $I' = \langle r_1, r_2, \ldots, r_{k+1} \rangle$, of consecutive requests in $I_1$, where $\mathbb{A}$ faults at most once. Since LRU faults on every request, they must be to $k+1$ different pages, $p_1, p_2, ..., p_{k+1}$. Let $p$ be the page requested immediately before $I'$. Clearly, $p$ must be in $\mathbb{A}$'s cache when it begins to process $I'$ (it is a paging algorithm). If $r_{k+1}$ is not a request to $p$, then $I'$ contains $k+1$ pages different from $p$, but at most $k-1$ of them are in $\mathbb{A}$'s cache when it begins to process $I'$ ($p$ is in its cache). Hence, $\mathbb{A}$ must fault at least twice on the requests in $I'$. On the other hand, if $r_{k+1}$ is a request to $p$, there are exactly $k$ requests in $I'$ which are different from $p$. At least one of them, say $p_i$, must cause a fault, since at most $k-1$ of them could have been in $\mathbb{A}$'s cache just before it began processing $I'$. If $\mathbb{A}$ faults on no other page than $p_i$ in $I'$, then all the pages $p, p_1, p_2, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k$ must be in $\mathbb{A}$'s cache just before it starts to process $I'$. Now, move the request to $p_i$ to the beginning of $I'$ which causes $\mathbb{A}$ to fault and evict one of the pages $p, p_1, p_2, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k$. Hence, it must

fault at least one additional time while processing the rest of this reordering of $I'$. □

Next, we show that LRU can never do significantly better than LRU-2. In order to show this, we need some definitions and lemmas characterizing LRU-2's behavior.

**Lemma 4.** *For any request sequence $I$, there exists a worst ordering of $I$ with respect to* LRU-2 *with all faults appearing before all hits.*

*Proof.* We describe how any permutation $I'$ of $I$ can be transformed, step by step, to a permutation $I_{\text{LRU-2}}$ with all hits appearing at the end of the sequence, without decreasing the number of faults LRU-2 will incur on the sequence. Let $I'$ consist of the requests $r_1, r_2, \ldots, r_n$, in that order.

If all hits in $I'$ appears after all the faults, we are done. Otherwise consider the first hit $r_i$ in $I'$ with respect to LRU-2. We construct a new ordering by moving $r_i$ later in $I'$.

Let $p$ denote the page requested by $r_i$. First, we remove $r_i$ from $I'$ and call the resulting sequence $I''$.

If LRU-2 never evicts $p$ in $I''$ or evicts $p$ at the same requests in $I''$ as it does in $I'$, then insert $r_i$ after $r_n$ in $I''$. This case is trivial since the behavior of LRU-2 on $I'$ and $I''$ is the same.

Thus, we need only consider the case where $p$ is evicted at some point after $r_{i-1}$ in $I''$, and is not evicted at the same point in $I'$. Let $r_j$, $j > i$, denote the first request causing $p$ to get evicted in $I''$ but not evicted in $I'$. Insert $r_i$ just after $r_j$ in $I''$. The resulting request sequence $I''$ is shown in Fig. 1 where $r_{p,1}$ and $r_{p,2}$ denote the next two requests to $p$ (if they exist).

$$I' : \langle \ldots, r_{i-1}, r_i, r_{i+1}, \ldots, r_j, \ldots, r_{p,1}, \ldots, r_{p,2}, \ldots \rangle$$
$$I'' : \langle \ldots, r_{i-1}, r_{i+1}, \ldots, r_j, r_i, \ldots, r_{p,1}, \ldots, r_{p,2}, \ldots \rangle$$

**Fig. 1.** The request sequence $I''$ after moving $r_i$

First note that moving a request to $p$ within the sequence only affects $p$'s position in the queue that LRU-2 evicts from. The relative order of the other pages stays the same. Just before $r_{i+1}$ the content of LRU-2's cache is the same for both sequences. Therefore, for $I''$, the behavior of LRU-2 is the same as for $I'$ until $p$ is evicted at $r_j$. Just after this eviction in $I''$, $p$ is requested by $r_i$ in $I''$. Thus, just before $r_{j+1}$, the cache contents are again the same for both sequences. This means that all pages that are in cache just before $r_{j+1}$, except $p$, are evicted no later for $I''$ than for $I'$. Hence, no faults are removed on requests to pages different from $p$, so we only need to count the faults removed on requests to $p$.

No faults on requests to $p$ are removed on requests after $r_{p,2}$ since after that request the second to last request to $p$ occurs at the same relative position in $I'$ as in $I''$, so LRU-2 cannot evict it in one and not the other. Hence, the only potential faults that could have been removed are at the two requests $r_{p,1}$ and $r_{p,2}$.

The only case that needs special care is the case where $r_{p,1}$ and $r_{p,2}$ both are faults in $I'$ but both are hits in $I''$. In all other cases at most one fault is removed which is counterbalanced by the fault created on $r_i$.

Consider the case where $r_{p,1}$ and $r_{p,2}$ both are faults in $I'$ but neither is in $I''$. First remove $r_{p,1}$ from $I''$ and call the resulting sequence $I'''$. Since $r_{p,2}$ is a fault in $I'$, $p$ must get evicted in the subsequence $\langle r_{p,1}, \ldots, r_{p,2} \rangle$ based on its second to last request in that subsequence, which is the same request for both $I'$ and $I'''$. Consequently, if $p$ is evicted in that subsequence of $I'$ it must also get evicted in that subsequence of $I'''$ and it follows that $r_{p,2}$ is a fault in both sequences and no faults have been removed.

The situation we are facing with $I'''$ is no different from the situation we faced with $I''$. We need to insert a (removed) request to $p$ (for $I''$ it was $r_i$ and for $I'''$ it is $r_{p,1}$) without removing any faults, except that we now have increased the number of faults among the first $j$ requests by at least one, and we have moved the problem of inserting a request to $p$ later in the sequence. We now proceed inductively on $I'''$ in the same manner as we did for $I''$ until we can insert the request to $p$ without removing any faults or we reach the end of the sequence (in which case we place it there).

Thus, we obtain $I_{\text{LRU-2}}$ in a finite number of steps. □

Thus, when considering a worst case sequence for LRU-2, one can assume that there is a prefix of the sequence containing all of the faults and no hits. In the remaining, we will only be considering such prefixes. We define LRU-2-phases, starting from any request in a request sequence $I$.

**Definition 3.** *Let* $I = \langle r_1, r_2, ..., r_n \rangle$ *be a request sequence for which* LRU-2 *faults on every request. The* LRU-2-phase starting at request $r_i$ *is* $P(r_i) = \langle r_i, r_{i+1}, ..., r_j \rangle$*, where* $j$ *is as large as possible under the restriction that no page should be requested three times in* $P(r_i)$*. A* LRU-2*-phase is* complete *if* $r_j$ *is not the last request in the* $I$*, i.e.,* $r_{j+1}$ *is a page which occurs twice in* $P(r_i)$*.*

**Lemma 5.** *Each complete* LRU-2*-phase contains at least* $2k + 1$ *requests to at least* $k + 1$ *distinct pages. In addition, it contains two requests to each of at least* $k$ *different pages.*

*Proof.* By definition, within a complete LRU-2-phase, $P$, there is a request to a page $p$ immediately after that phase. This request causes a fault, and $p$ was requested at least twice within the phase. In order for $p$ to be evicted within the phase $P$, the second to last request to each of the $k-1$ other pages in cache must have occurred more recently than the first request to $p$ in $P$. Thus, counting $p$, at least $k$ distinct pages must have been requested twice in $P$. In addition, the page causing the second eviction of $p$ within $P$ cannot have been in cache at that point, so $P$ consists of at least $2k + 1$ requests. □

**Lemma 6.** *Let* $p$ *be the page that starts a complete* LRU-2*-phase containing exactly* $2k+1$ *requests, then the following phase (if it exists) starts with a request to* $p$*.*

*Proof.* In general after a complete LRU-2-phase, LRU-2 has at least $k-1$ of the pages requested twice in that phase in cache. If the phase ends with the second request to a page, then LRU-2 contains $k$ of the pages requested twice. This fact follows from the observation that by the LRU-2 policy no page with two requests in a phase can get evicted if there is a page in cache with only one request in that phase.

This means that a phase containing only $2k+1$ requests must end with a request to the only page with one request in that phase. Before that request the cache contains $k$ pages which have all been requested twice in the current phase, hence $p$ is the page with the earliest second request. This means $p$ gets evicted on the last request in the phase and hence (by the construction of LRU-2-phases) it must be the page starting the next phase. □

By induction the above shows that if there exist several consecutive phases, each containing $2k+1$ requests, then they must all begin with a request to the same page. This then shows that if $p_1, p_2, \ldots, p_k$ are the $k$ pages requested twice in a phase containing $2k+1$ requests, then after the first request in the following phase (if it exists) all of the pages $p_1, p_2, \ldots, p_k$ are in LRU-2's cache.

**Lemma 7.** *For any sequence $I$ of page requests,*

$$\text{LRU-2}_W(I) \leq (1 + \frac{1}{2k+2})\text{LRU}_W(I).$$

*Proof.* Consider any sequence $I$ of requests. By Lemma 4, there exists a worst permutation, $I_{\text{LRU-2}}$, of $I$ such that LRU-2 faults on each request of a prefix $I_1$ of $I_{\text{LRU-2}}$ and on no requests after $I_1$. Partition $I_1$ into LRU-2-phases. We will now inductively transform $I_1$ into a sequence $I_1'$ such that

$$\text{LRU-2}(I_1) \leq (1 + \frac{1}{2k+2})\text{LRU}_W(I_1').$$

Start at the beginning of $I_1$, and consider the LRU-2-phase starting with the first request not already placed in a processed phase. By Lemma 6 each LRU-2-phase contains at least a total of $2k+1$ requests to at least $k+1$ distinct pages. Since each page requested in a LRU-2-phase is at most requested twice, a LRU-2-phase containing at least $2k+2$ requests can be partitioned into two sets, each containing at least $k+1$ pages, none of which are repeated. Each of these sets of requests can then be ordered so that LRU faults on every request.

Hence, suppose the current LRU-2-phase contains exactly $2k+1$ requests. See Fig. 2 where | marks the beginning of a new phase in $I_1$ which contains exactly $2k+1$ requests. Let $p_1, p_2, \ldots, p_k$ be the $k$ pages requested twice and $q_1$ be the page requested once in that phase and let $p_1$ be the page which begin the following phase. The request $r_i$ to $p_1$ which starts the following phase must evict $q_1$ and hence all the pages $p_1, p_2, \ldots, p_k$ are in LRU-2's cache just before the request to $r_{i+1} = q_2$. It follows that $q_2 \notin \{p_1, p_2, \ldots, p_k\}$.

$$\langle \ldots, |p_1, \ldots, q_1, |r_i = p_1, r_{i+1} = q_2, r_{i+2} \ldots \rangle$$

**Fig. 2.** A LRU-2-phase containing $2k + 1$ requests

By moving $r_i$ to the end of the request sequence it follows from the above that the modified phase in question now contains at least $2(k+1)$ requests (the $2k+1$ requests and the request to $q_2$) and by the same argument as above it follows that it is possible to make LRU fault on every request. Hence in each such phase LRU faults on (possibly) one request less than LRU-2. The next LRU-2-phase to be processed starts with $r_{i+2}$ or later.

Let $l$ denote the total number of modified phases. For each modified phase $i$, there are $s_i \geq 2(k+1)$ requests, plus possibly one additional request which LRU-2 faulted on and has been moved to the end. Thus, LRU faults at least $\sum_{i=1}^{l} s_i$ times and LRU-2 faults at most $\sum_{i=1}^{l}(s_i + 1)$ times. It follows that

$$\begin{aligned} \text{LRU-2}_W(I) &\leq \frac{\sum_{i=1}^{l}(s_i+1)}{\sum_{i=1}^{l} s_i}\text{LRU}_W(I) \\ &\leq \frac{l(2k+3)}{l(2k+2)}\text{LRU}_W(I) \\ &= (1+\frac{1}{2k+2})\text{LRU}_W(I) \end{aligned}$$

□

Combining Theorem 2 and the lemma above gives the following:

**Theorem 4.** LRU-2 *and* LRU *are* $(1+\frac{1}{2k+2}, \frac{k+1}{2})$*-related, i.e., they are asymptotically comparable in* LRU-2*'s favor.*

## 5 Concluding Remarks

In contrast to the results using competitive analysis, relative worst order analysis yields a theoretical justification for superiority of LRU-2 over LRU, confirming previous empirical evidence. It would be interesting to see if these results generalize to LRU-K for $K > 2$. Recently, we have shown that the competitive ratio for LRU-K is $kK$, and that the separation result showing that LRU-K can be better than LRU holds. The question is: Does the asymptotic comparability still hold.

Although it was shown here that LRU-2 and LRU are asymptotically comparable, it would be interesting to know if the stronger result, that LRU-2 and LRU are comparable using relative worst order analysis, holds. If they are, then the above results show that the relative worst order ratio of LRU to LRU-2 is $\frac{k+1}{2}$.

Note that any result showing that the relative worst order ratio is defined for two algorithms immediately gives a result showing that they are asymptotically comparable. Thus, the results from [5], showing that LRU is at least as good as any conservative algorithm and better than Flush-When-Full (FWF), combined with the results proven here, show that LRU-2 is asymptotically comparable to any conservative algorithm and FWF, in LRU-2's favor in each case.

An algorithm called RLRU was proposed in [5] and shown to be better than LRU using relative worst order analysis. We conjecture that LRU-2 is also asymptotically comparable to RLRU in LRU-2's favor. We have found a family of sequences showing that LRU-2 can be better than RLRU, but would also like to show that the algorithms are asymptotically comparable.

## Acknowledgments

The authors would like to thank Peter Sanders for bringing LRU-2 to their attention. The second author would like to thank Troels S. Jensen for helpful discussions.

## References

1. Susanne Albers. Online algorithms: A survey. In *Proceedings of the 18th International Symposium on Mathematical Programming*, pages 3–26, 2003.
2. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
3. Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive Paging with Locality of Reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995.
4. Joan Boyar and Lene M. Favrholdt. The relative worst order ratio for on-line algorithms. In *Proceedings of the Fifth Italian Conference on Algorithms and Complexity*, volume 2653 of *Lecture Notes in Computer Science*, pages 58–69. Springer-Verlag, 2003. Extended version to appear in *ACM Transactions on Algorithms*.
5. Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. The Relative Worst Order Ratio Applied to Paging. In *Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 718–727. ACM Press, 2005.
6. Joan Boyar and Paul Medvedev. The Relative Worst Order Ratio Applied to Seat Reservation. In *Proceedings of the Nineth Scandinavian Workshop on Algorithm Theory*, volume 3111 of *Lecture Notes in Computer Science*, pages 90–101. Springer-Verlag, 2004.
7. Leah Epstein, Lene M. Favrholdt, and Jens S. Kohrt. Separating Scheduling Algorithms with the Relative Worst Order Ratio. *Journal of Combinatorial Optimization*. To appear.
8. Amos Fiat and Ziv Rosen. Experimental Studies of Access Graph Based Heuristics: Beating the LRU Standard? In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 63–72, 1997.
9. Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive Snoopy Caching. *Algorithmica*, 3:79–119, 1988.

10. Jens Svalgaard Kohrt. *Online Algorithms under New Assumptions.* PhD thesis, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark, 2004.
11. Sven Oliver Krumke, Willem de Paepe, Jörg Rambau, and Leen Stougie. Online Bin Coloring. In *Proceedings of the Nineth Annual European Symposium on Algorithms*, volume 2161 of *Lecture Notes in Computer Science*, pages 74–85. Springer-Verlag, 2001.
12. Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 297–306, 1993.
13. Daniel D. Sleator and Robert E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28(2):202–208, 1985.
14. Neal Young. The $k$-Server Dual and Loose Competitiveness for Paging. *Algorithmica*, 11(6):525–541, 1994.

# Improved Approximation Bounds for Edge Dominating Set in Dense Graphs

Jean Cardinal, Stefan Langerman*, and Eythan Levy

Computer Science Department
Université Libre de Bruxelles, CP212
B–1050 Brussels, Belgium
{jcardin,slanger,elevy}@ulb.ac.be

**Abstract.** We analyze the simple greedy algorithm that iteratively removes the endpoints of a maximum-degree edge in a graph, where the degree of an edge is the sum of the degrees of its endpoints. This algorithm provides a 2-approximation to the minimum edge dominating set and minimum maximal matching problems. We refine its analysis and give an expression of the approximation ratio that is strictly less than 2 in the cases where the input graph has $n$ vertices and at least $\epsilon\binom{n}{2}$ edges, for $\epsilon > 1/2$. This ratio is shown to be asymptotically tight for $\epsilon > 1/2$.

## 1 Introduction

While there exist sophisticated methods yielding approximate solutions to many NP-hard combinatorial optimization problems, the methods that are the simplest to implement are often the most widely used. Among these methods, greedy strategies are extremely popular and certainly deserve thorough analyses.

We study the worst-case approximation factor of a simple greedy algorithm for the following two NP-hard problems.

**Definition 1** (Minimum Edge Dominating Set)
INPUT: *A graph* $G = (V, E)$.
SOLUTION: *A subset* $M \subseteq E$ *of edges such that each edge in* $E$ *shares an endpoint with some edge in* $M$.
MEASURE: $|M|$.

**Definition 2** (Minimum Maximal Matching)
INPUT: *A graph* $G = (V, E)$.
SOLUTION: *A subset* $M \subseteq E$ *of disjoint edges such that each edge in* $E$ *shares an endpoint with some edge in* $M$.
MEASURE: $|M|$.

It has been noted since long ago that Minimum Edge Dominating Set (EDS) and Minimum Maximal Matching (MMM) admit optimal solutions of the same size and that an optimal solution to EDS can be transformed in polynomial

* Chercheur qualifié du FNRS.

T. Erlebach and C. Kaklamanis (Eds.): WAOA 2006, LNCS 4368, pp. 108–120, 2006.

time into an optimal solution to MMM [13], the converse transformation being trivial.

The algorithm that we analyze in this paper uses the degree of the edges, with the degree of an edge being the sum of the degrees of its endpoints. It iteratively removes the highest-degree edge and updates the graph accordingly, as shown in Algorithm 1. The algorithm returns a maximal matching, which provides a

**Algorithm 1.** The greedy algorithm

```
res ← ∅
while E(G) ≠ ∅ do
   e ← arg max_{e∈E(G)} deg_G(e)
   res ← res ∪ {e}
   for each edge f adjacent to e do
      E(G) ← E(G)\{f}
   end for
   E(G) ← E(G)\{e}
end while
return res
```

solution to both our problems. The algorithm therefore guarantees exactly the same approximation ratios for the two problems.

It is well-known that any maximal matching $M$ provides a 2-approximation for MMM, as each edge in the optimal solution can cover at most two edges of $M$. Our algorithm is thus clearly a 2-approximation algorithm and is expected to return small matchings as the greedy step always selects a high-degree edge. We however refine this analysis, and provide a tight approximation factor as a function of the density of the graph.

### Our Contributions

We provide a new bound on the approximation ratio of the greedy heuristic for our problems in graphs with at least $\epsilon\binom{n}{2}$ edges ($\epsilon$-dense graphs). This bound is asymptotic to $1/(1-\sqrt{(1-\epsilon)/2})$, which is smaller than 2 when $\epsilon$ is greater than $1/2$. We further provide a family of tight examples for our bound. No algorithm for $\epsilon$-dense graphs with a better approximation ratio than the one shown in this paper seems to be known.

### Related Works

The MMM and EDS problems go back a long way. Both problems are already referred to in the classical work of Garey and Johnson [6] on NP-completeness. Yannakakis and Gavril [13] then showed that EDS remains NP-hard when restricted to planar or bipartite graphs of maximum degree 3, and gave a polynomial-time algorithm for MMM in trees. Later, Horton et al. [8] and Srinivasan et al. [12] gave

additional hard and polynomially solvable classes of graphs. More recently, Carr et al. [2] gave a $2\frac{1}{10}$-approximation algorithm for the weighted edge dominating set problem, a result which was later improved to 2 by Fujito et al. [5]. Finally, Chlebìk and Chlebìkovà [3] showed that it is NP-Hard to approximate EDS(and hence also MMM) within any factor better than 7/6.

Another recent trend of research on approximation algorithms deals with expressing approximation ratios as functions of some density parameters [4,7,9], related to the number of edges, or the minimum and maximum degrees. Not many such results have yet been obtained for our problems. It was nevertheless shown in [1] that MMM and EDS are approximable within ratios that are asymptotic to $\min\{2, 1/(1-\sqrt{1-\epsilon})\}$ for graphs having at least $\epsilon\binom{n}{2}$ edges, and to $\min\{2, 1/\epsilon\}$ for graphs having minimum degree at least $\epsilon n$.

## 2 Analysis of Algorithm 1

### Definitions and Notations

Let $G = (V, E)$ be a (simple, loopless, undirected) graph, with $V = \{v_1, \ldots, v_n\}$. Let $OPT$ be a fixed optimal solution to MMM in $G$ and let $T$ be the set of endpoints in $OPT$. Let $M = \{e_1, \ldots, e_\mu\}$ be a set of $\mu$ edges returned by an execution of the greedy algorithm on $G$. We assume that these edges are ordered according to the order in which they were chosen by the algorithm.

The definition of the algorithm ensures that $M$ is a maximal matching. Since $M$ is a matching, at least one endpoint of each edge $e_i$ belongs to $T$. Let us call $\{v_1, \ldots, v_{2\mu}\}$ the endpoints of the edges of $M$, with $e_i = v_{2i-1}v_{2i}$ and $v_{2i-1} \in T$. Since the matching $M$ is maximal, the set of vertices $\{v_{2\mu+1}, \ldots, v_n\}$ forms a stable set, i.e. a set of vertices sharing no edge. The set of vertices $V \backslash T$ also forms a stable set as the vertices in $T$ are the endpoints of a maximal matching. Fig. 1 shows an example with $\mu = 6$ and $|OPT| = 5$. Our assumptions on the ordering of the vertices ensure that a vertex has a higher index when it is included later (or never) in the heuristic solution and that the vertex with lowest index in $e_i$ belongs to $T$.

As can be seen in Fig. 1, there are two types of edges in $M$. Edges of the first type have only one endpoint in $T$. We let $X$ be the set of these endpoints. Edges of the second type have both endpoints in $T$. Let $a$ be the number of such edges. Let finally $b$ be the number of vertices of $T$ outside $M$. Fig. 1 also illustrates $X$, $a$ and $b$. Note that in practice the two types of edges can be interleaved in $M$, whereas they are shown separated in the figure for the sake of clarity.

The approximation ratio is $\beta = \mu/|OPT|$. This quantity is fixed when $M$ and $OPT$ are given. In order to give an upper bound on $\beta$, we prove an upper bound on the number of edges in a graph when $M$ and $OPT$ are fixed. This bound is then inverted in order to obtain an upper bound on $\beta$ as a function of the number of edges. Our results are expressed in terms of the *density* of our graphs, according to the following definitions. We define an *$\epsilon$-dense graph* as a graph with at least $\epsilon\binom{n}{2}$ edges.

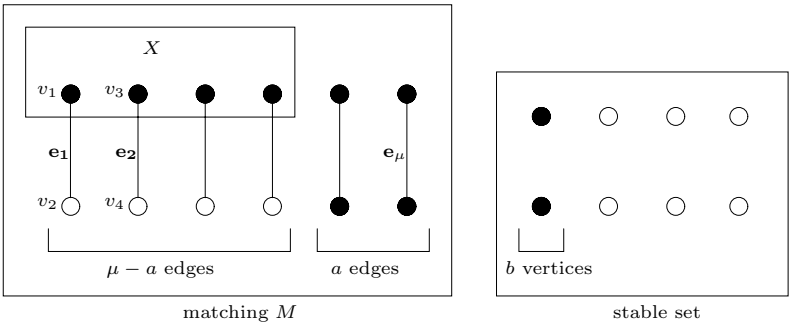



**Fig. 1.** An example with $\mu = 6$ and $|OPT| = 5$. Black vertices are the endpoints of the minimum maximal matching.

The following additional graph-theoretic notations will be useful. For any vertex set $W \subseteq V$ and vertex $v$, let $N_W(v)$ be the set of neighbors of $v$ in set $W$ and let $d_W(v) = |N_W(v)|$. Let an *anti-edge* $xy$ be a pair of vertices $x$ and $y$ sharing no edge. Let $N_W^<(v_j)$ be the set of neighbors $v_i$ of $v_j$ with $i < j$ and $v_i \in W$, and let $d_W^<(v_j) = |N_W^<(v_j)|$. For any of these notations, the subscript $W$ may be omitted when $W = V$. We also use the classical notation $G[X]$ for the subgraph of $G$ induced by a vertex set $X$. Let $\bar{m}(G) = \binom{n}{2} - m(G)$ be the number of anti-edges in $G$. We omit the parameter $G$ when it is clear from context. We define $G \times G'$, the *join* of graphs $G = (V, E)$ and $G' = (V', E')$ as a new graph that contains all the vertices and edges of $G$ ang $G'$ as well as all the possible edges joining both sets of vertices.

### Upper Bound

Lemma 1 shows that a certain set of vertices has degree at most $|T|$. This result is then used by Lemma 2 in order to find an upper bound on the number of edges in the graph.

**Lemma 1.** *If $d_X^<(v_j) > 0$ for some vertex $v_j$, then $d(v_j) \leq |T|$.*

*Proof.* We call the vertices of $T$ black vertices and the vertices outside of $T$ white vertices. Let $i$ be the smallest index such that $v_i \in X$ and $v_i v_j \in E$. Let $V_a^b = (v_a \dots v_b)$. Fig. 2 illustrates these notations. We can express the degree of $v_j$ as:

$$d(v_j) = d_{V_1^{i-1}}(v_j) + d_{V_i^n}(v_j).$$

Since $v_j$ has no neighbor in $V_1^{i-1} \cap X$, we have $d_{V_1^{i-1}}(v_j) \leq |V_1^{i-1} \backslash X|$ and therefore

$$d(v_j) \leq |V_1^{i-1} \backslash X| + d_{V_i^n}(v_j). \tag{1}$$

It can easily be seen that $|V_1^{i-1} \backslash X| = |V_1^{i-1} \cap T|$ and therefore

$$d(v_j) \leq |V_1^{i-1} \cap T| + d_{V_i^n}(v_j).$$

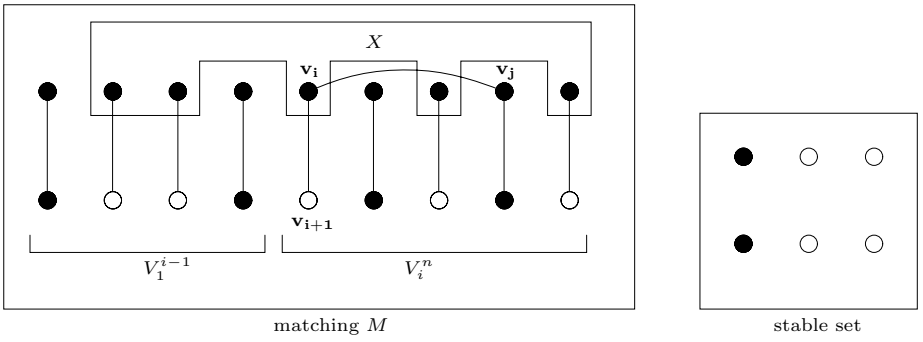



**Fig. 2.** Structure of the matching $M$. In this example, $v_j$ was chosen inside the matching and outside $X$. Note that Lemma 1 also allows $v_j$ to be in the stable set or in $X$.

The greedy algorithm ensures that edge $v_i v_{i+1}$ has maximum degree in $G[V_i^n]$, and therefore

$$d(v_j) \leq |V_1^{i-1} \cap T| + d_{V_i^n}(v_{i+1}). \tag{2}$$

It is worth noticing that this is the only place in the whole proof of Theorem 1 where this property is used. Finally, since $v_{i+1}$ is a white vertex, it can only be adjacent to vertices in $T$, as the white vertices form a stable set. Therefore

$$\begin{aligned} d(v_j) &\leq |V_1^{i-1} \cap T| + |V_i^n \cap T| \\ &= |T|. \end{aligned}$$

□

The following result provides a lower bound on the number of anti-edges in the graph, hence an upper bound on the number of edges. Its proof uses counting arguments that heavily rely on the bound given in Lemma 1. Recall that $a$ is the number of edges of $M$ having both endpoints in $T$, and that $b$ is the number of vertices of $T$ that are outside $M$.

**Lemma 2**

$$\bar{m} \geq 2\binom{n/2 - a - b}{2}$$

*Proof.* Let $\bar{d}_W(v)$, the *anti-degree* of $v$, be the number of anti-edges between $v$ and vertices of $W$. Thus

$$\bar{d}_W(v) = \begin{cases} |W| - d_W(v) & \text{if } v \notin W \\ |W| - 1 - d_W(v) & \text{otherwise.} \end{cases}$$

We first define a family of vertex sets $\{X_i\}$ and show a lower bound on $\bar{m}$ as a function of the sizes of these sets. We call the vertices in (resp. outside) $T$ *black* (resp. *white*) vertices.

The sets of vertices are the following (see Fig. 3): $X_1$ and $X_2$ are defined as the black and white endpoints of $\mu - a - b$ arbitrary black-white edges of $M$. Sets $X_3$ and $X_4$ are obtained by splitting the $b$ black vertices outside of $M$ into two sets of equal sizes (rounding if necessary). Sets $X_5$ and $X_6$ are obtained by splitting the $n - 2\mu - b$ white vertices outside of $M$ into two sets of equal sizes. Finally, $X_9$ and $X_{10}$ are obtained by dividing the remaining $b$ vertices of the matching into sets of equal sizes. We define $x_i = |X_i|$ for each set $X_i$.

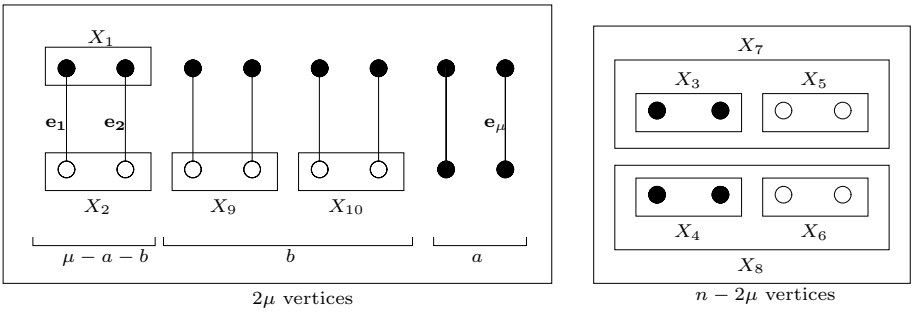



**Fig. 3.** Notations for the vertex sets

We first show

$$\bar{m} \geq \binom{x_1}{2} + \binom{x_2}{2} + \binom{x_7}{2} + \binom{x_8}{2} + x_2x_9 + x_2x_5 + x_2x_{10} + x_2x_6 \qquad (3)$$

Note that each set $X_i$ except $X_1$ is stable, because it either contains only white vertices or only vertices outside $M$. This explains the second,third and fourth terms in the above sum. For each term of the form $x_ix_j$ in the sum, both $X_i$ and $X_j$ contain only white vertices, and therefore share no edge, since any set of white vertices in $G$ is stable. Note that no anti-edge is counted twice, since our anti-edges involve vertices taken in and between disjoint vertex sets.

Concerning the additional number of $\binom{x_1}{2}$ anti-edges required, we use Lemma 1 to prove that every edge inside $X_1$ is compensated for by an anti-edge between a vertex in $X_1$ and a vertex outside $X_1$. For each $v_j \in X_1$, we have:

$$\begin{aligned} d^{<}_{X_1}(v_j) &\leq d_{X_1}(v_j) \\ &= d(v_j) - d_{V \setminus X_1}(v_j). \end{aligned}$$

Applying Lemma 1 yields:

$$d^{<}_{X_1}(v_j) \leq |T| - d_{V \setminus X_1}(v_j).$$

Using $|T| = \mu + a + b$ and $\mu \leq n/2$ yields

$$d^{<}_{X_1}(v_j) \leq n - (\mu - a - b) - d_{V \setminus X_1}(v_j).$$

Finally, since $|V \backslash X_1| = n - (\mu - a - b)$, the definition of the anti-degree yields

$$d^{<}_{X_1}(v_j) \leq \bar{d}_{V \backslash X_1}(v_j).$$

We now take sums over the elements of $X_1$:

$$\sum_{v_j \in X_1} d^{<}_{X_1}(v_j) \leq \sum_{v_j \in X_1} \bar{d}_{V \backslash X_1}(v_j).$$

Since the sets $N^{<}_{X_1}(v_j)$ corresponding to the values $d^{<}_{X_1}(v_j)$ in the above sum form a partition of the edges of $G[X_1]$, we have

$$m(G[X_1]) \leq \sum_{v_j \in X_1} \bar{d}_{V \backslash X_1}(v_j).$$

From the definition of $\bar{m}$, we have:

$$\binom{x_1}{2} \leq \bar{m}(G[X_1]) + \sum_{v_j \in X_1} \bar{d}_{V \backslash X_1}(v_j).$$

The above relation thus implies the existence of at least $\binom{x_1}{2}$ anti-edges involving vertices of $X_1$.

There remains to show that bound 3 is greater than $2\binom{n/2-a-b}{2}$. Plugging $x_2 = x_1$, $x_9 = x_3$, and $x_{10} = x_4$ into 3 yields:

$$\bar{m} \geq \binom{x_1}{2} + \binom{x_2}{2} + \binom{x_7}{2} + \binom{x_8}{2} + x_1x_3 + x_1x_5 + x_2x_4 + x_2x_6.$$

and therefore

$$\bar{m} \geq \binom{x_1}{2} + \binom{x_2}{2} + \binom{x_7}{2} + \binom{x_8}{2} + x_1x_7 + x_2x_8.$$

The desired result follows from repeated applications of the relation $\binom{x+y}{2} = \binom{x}{2} + \binom{y}{2} + xy$ :

$$\begin{aligned}
\bar{m} &\geq \binom{x_1}{2} + \binom{x_2}{2} + \binom{x_7}{2} + \binom{x_8}{2} + x_1(x_7) + x_2(x_8) \\
&= \binom{|X_1 \cup X_7|}{2} + \binom{|X_2 \cup X_8|}{2} \\
&= \binom{x_1 + x_7}{2} + \binom{x_2 + x_8}{2} \\
&= \binom{\lfloor n/2 - a - b \rfloor}{2} + \binom{\lceil n/2 - a - b \rceil}{2} \\
&=^{*} \begin{cases} 2\binom{n/2-a-b}{2} & \text{if } n \text{ is even} \\ 2\binom{n/2-a-b}{2} + 1/4 & \text{otherwise.} \end{cases} \\
&\geq 2\binom{n/2 - a - b}{2}.
\end{aligned}$$

□

Theorem 1 is essentially a consequence of this upper bound on the number of edges.

**Theorem 1.** *The approximation ratio of the greedy heuristic in $\epsilon$-dense graphs with $n$ vertices is at most*

$$\begin{cases} 2 & \text{if } \epsilon \leq \frac{1}{2} + \frac{1}{n-1} \\ \left[1 - \frac{1}{2n} - \sqrt{\frac{1}{4n^2} + \left(1 - \frac{1}{n}\right)\frac{(1-\epsilon)}{2}}\right]^{-1} & \text{otherwise.} \end{cases}$$

$$\longrightarrow_{n\to\infty} \begin{cases} 2 & \text{if } \epsilon \leq \frac{1}{2} \\ \left[1 - \sqrt{\frac{1-\epsilon}{2}}\right]^{-1} & \text{otherwise.} \end{cases}$$

*Proof.* We know from Lemma 2 that $\bar{m} \geq 2\binom{n/2-a-b}{2}$. Simple algebra using $\beta = \mu/|OPT|$, $2|OPT| = \mu + a + b$ and $\mu \leq n/2$ implies

$$a + b \leq \frac{n}{2}\left[\frac{2-\beta}{\beta}\right].$$

and therefore

$$\bar{m} \geq 2\binom{n/2 - \frac{n}{2}\left[\frac{2-\beta}{\beta}\right]}{2} = 2\binom{n\left(\frac{\beta-1}{\beta}\right)}{2}. \tag{4}$$

Let $x = (\beta - 1)/\beta$. We would like to express the above inequality as an upper bound on $\beta$, i.e. on $x$. The inequality can now be written as

$$f(x) = n^2x^2 - nx - \bar{m} \leq 0.$$

Differentiating $f$ with respect to $x$ shows that $f$ decreases when $x < \frac{1}{2n}$ and increases when $x > \frac{1}{2n}$. The value of $f(x)$ can therefore only be negative when $x^- \leq x \leq x^+$, where $x^-$ and $x^+$ are the roots of $f(x)$. Solving the second-order equation $f(x) = 0$ yields

$$x^- = \frac{1}{2n} - \sqrt{\frac{1}{4n^2} + \frac{\bar{m}}{n^2}}$$

and

$$x^+ = \frac{1}{2n} + \sqrt{\frac{1}{4n^2} + \frac{\bar{m}}{n^2}}.$$

The value of $x^-$ is always negative and thus $x^- \leq x$ brings us no additional knowledge on the ratio. Rewriting inequality $x \leq x^+$ yields

$$\frac{\beta - 1}{\beta} \leq \frac{1}{2n} + \sqrt{\frac{1}{4n^2} + \frac{\bar{m}}{n^2}}$$

and

$$\beta \leq \left[1 - \frac{1}{2n} - \sqrt{\frac{1}{4n^2} + \frac{\bar{m}}{n^2}}\right]^{-1}.$$

Reverting to $m$ and setting $m \geq \epsilon\binom{n}{2}$ yields the desired result

$$\beta \leq \left[1 - \frac{1}{2n} - \sqrt{\frac{1}{4n^2} + \left(1 - \frac{1}{n}\right)\frac{(1-\epsilon)}{2}}\right]^{-1}$$
$$= \left[1 - O\left(\frac{1}{n}\right) - \sqrt{\frac{1-\epsilon}{2} + O\left(\frac{1}{n}\right)}\right]^{-1}.$$

Direct algebraic manipulations show that

$$\left[1 - \frac{1}{2n} - \sqrt{\frac{1}{4n^2} + (1-\epsilon)\left(\frac{1}{2} - \frac{1}{2n}\right)}\right]^{-1} < 2$$
$$\iff \epsilon > \frac{1}{2}\left(\frac{n}{n-1}\right)$$
$$\iff \epsilon > \frac{1}{2} + \frac{1}{n-1}.$$

□

**Tightness**

**The case $\epsilon \geq 7/9$.** Let $\zeta_{n,k} = K_{n-2k} \times K_{k,k}$, where $K_{n-2k}$ is a complete graph with $n - 2k$ vertices and $K_{k,k}$ a complete bipartite graph with two stable sets of size $k$ (see Fig. 4(b) for an example). Such a graph can be compared with the *complete split graph* $\Psi_{n,k}$ (see Fig. 4(a)), which is defined as the join of a clique of size $n - k$ and an independent set of size $k$ and is a tight example for the simpler greedy algorithm analyzed in [1].

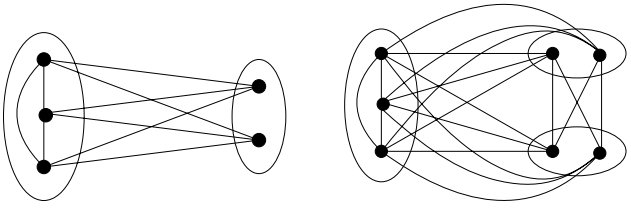
Algorithm 1 always finds a perfect matching in $\zeta_{n,k}$. On the other hand, the following matching is clearly maximal: match $k$ vertices of the clique with $k$ vertices of one independent set, and match the remaining vertices of the clique among themselves. This is always possible when $k$ and $n$ are even and $k \leq n/3$ and yields a matching of size $(n-k)/2$. Therefore we have the following bound on the approximation ratio: $\beta = \mu/|OPT| \geq n/(n-k)$.

The number of edges of $\zeta_{n,k}$ is given by $m = \binom{n}{2} - 2\binom{k}{2}$ and therefore $k = \left(1 + \sqrt{1 + 4\left[\binom{n}{2} - m\right]}\right)/2$. We denote by $\epsilon$ the ratio $m/\binom{n}{2}$, i.e. the density of $\zeta_{n,k}$. From the above equality, we have $k = \left(1 + \sqrt{1 + 4\binom{n}{2}(1-\epsilon)}\right)/2$.

Plugging this equation into the inequality for $\beta$ above yields

$$\beta \geq \left[1 - \frac{1}{2n} - \sqrt{\frac{1}{4n^2} + \left(1 - \frac{1}{n}\right)\frac{(1-\epsilon)}{2}}\right]^{-1},$$

which matches the upper bound on the ratio obtained in Theorem 1. Plugging the condition $k \leq n/3$ into $m = \binom{n}{2} - 2\binom{k}{2}$ yields $\epsilon \geq 7/9 + O(1/n)$. The graphs $\zeta_{n,k}$ with $n$ and $k$ even are thus a collection of tight examples for our bound when $\epsilon \geq 7/9$.



(a) Complete split graph $\Psi_{5,2}$ (b) Tight example $\zeta_{7,2}$

**Fig. 4.** Tight examples

**The general case.** A slightly more intricate family of graphs can be built, which provide a collection of asymptotically tight examples for our ratio for any $\epsilon \geq 1/2$. We first describe the special case when $\epsilon = 1/2$ and $\beta \to 2$. The graph is the following (see Fig. 5) :

$$B \equiv M_{k/2} \times I_k \times K_1$$

where $M_{k/2}$ is a matching of $k/2$ edges, $I_k$ an empty graph with $k$ vertices, and $K_1$ is an isolated vertex.

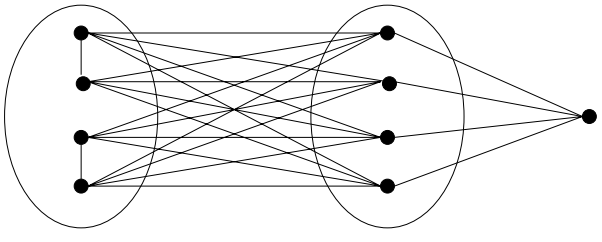


**Fig. 5.** Tight example $A_{9,4}$

It is easy to see that at each step of Algorithm 1 there exists an edge between the matching and the stable set that has maximum degree. Therefore the algorithm might choose $k$ of these edges thus obtaining a cover of size $k$. On the other hand, taking all the edges of $M_{k/2}$ and an additional edge incident to $K_1$ yields a cover of size $k/2 + 1$. Therefore $\beta \geq k/(k/2 + 1)$ which tends to 2 as $k$ tends to infinity. It is further straightforward to check that the density of this graph is $1/2 + O(1/n)$.

For other values of $\epsilon$, we generalize the above example by joining it to a clique, i.e. we build the following general family:

$$A_{n,k} \equiv (M_{k/2} \times I_k \times K_1) \times K_{n-2k-1}$$

Note that $A_{n,k}$ is well-defined for any odd $n$ and even $0 \leq k < n/2$ and that the limiting values of $k$ correspond respectively to $B$ and to $K_n$. The density

of $A_{n,k}$ therefore spans the whole range $]1/2, 1]$. It is further easy to check that $m(A_n, k) = \binom{n}{2} - \binom{k}{2} + O(n)$.

In $A_{n,k}$, Algorithm 1 will first empty the clique $K_{n-2k-1}$, leaving us again with a subgraph isomorphic to $B$. The algorithm can therefore return a matching of size $(n-2k-1)/2+k = (n-1)/2$. On the other hand, taking a perfect matching in the clique $K_{n-2k-1}$ together with the same $k/2+1$ edges described above for $B$ yields a cover of size $(n-2k-1)/2 + k/2 + 1 = (n-k+1)/2$. Therefore we have

$$\beta \geq \frac{(n-1)/2}{(n-k+1)/2} = \frac{n-1}{n-k+1}$$

Setting $m(A_n, k) = \binom{n}{2} - \binom{k}{2} + O(n)$ and $\epsilon = m/\binom{n}{2}$ as for $\zeta_{n,k}$ yields

$$\frac{n-1}{n-k+1} \to \left[1 - \sqrt{\frac{1-\epsilon}{2}}\right]^{-1} \quad \text{as } n \to \infty$$

Our lower bound on the ratio thus asymptotically matches the upper bound of Theorem 1. Note that we have made no special assumption on $k$ as we had done for $\zeta_{n,k}$. The graphs $A_{n,k}$ with odd $n$ and even $0 \leq k < n/2$ are therefore a family of asymptotically tight graphs for $\epsilon \geq 1/2$. Asymptotically tight examples for even $n$ may be obtained by slight adaptation of the above graphs.

## 3 Conclusion

Several variants to Algorithm 1 could be devised. For example, one could decide to slightly alter Algorithm 1 by each time selecting the edge that has the highest degree in the original graph rather than the updated graph. This variant is interesting as it can easily be implemented in time $O(n + m)$ using counting sort. Another interesting variant is the one in which one does not select the highest degree edge, but rather the edge defined by the highest degree vertex and its highest degree neighbor. We claim that Theorem 1 remains valid for these two variants. One should first notice that the only place in our analysis where explicit use is made of the strategy for choosing an edge is in Lemma 1. It is almost straightforward to adapt its proof for both variants.

Further, it can be checked that the asymptotic bound of $1/\epsilon$ for graphs with minimum degree at least $\epsilon n$ obtained for the maximal matching heuristic in [1] is also tight for Algorithm 1, with the same tight examples as those described in section 2.

Finally, Algorithm 1 also provides a 2-approximation for Minimum Vertex Cover by taking the endpoints of the maximal matching returned by the algorithm. The ratio obtained in Theorem 1 is also valid for this problem by slight adaptations to the proofs. The analytical form of our asymptotic result compares interestingly with that of both the simplest [1] and the best known approximation algorithm for Minimum Vertex Cover in $\epsilon$-dense graphs [10] : $1/(1 - \sqrt{(1-\epsilon)/2})$ against $1/(1 - \sqrt{(1-\epsilon)})$ and $1/(1 - \sqrt{(1-\epsilon)/4})$. Fig. 6 compares these ratios.

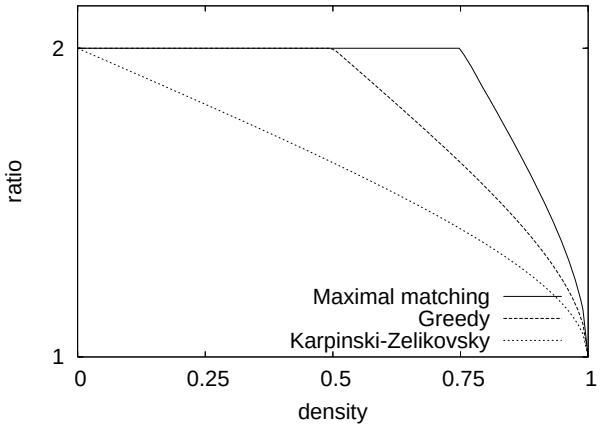



**Fig. 6.** A comparison of the ratios provided by the maximal matching heuristic, the greedy algorithm and Karpinski and Zelikovsky's algorithm

**Acknowledgments.** The authors wish to thank Martine Labbé, with whom this research was initiated, and Hadrien Mélot, author of the GraPHedron software [11], which was used to formulate the initial conjectures.

## References

1. J. Cardinal, M. Labbé, S. Langerman, E. Levy, and H. Mélot. A tight analysis of the maximal matching heuristic. In *Proc. of The Eleventh International Computing and Combinatorics Conference (COCOON)*, LNCS, pages 701–709. Springer–Verlag, 2005.
2. R. Carr, T. Fujito, G. Konjevod, and O. Parekh. A 2 1/10-approximation algorithm for a generalization of the weighted edge-dominating set problem. *Journal of Combinatorial Optimization*, 5:317–326, 2001.
3. M. Chlebìk and J. Chlebìkovà. Approximation hardness of edge dominating set problems. *Journal of Combinatorial Optimization*, 11(3):279–290, 2006.
4. A.V. Eremeev. On some approximation algorithms for dense vertex cover problem. In *Proc. of SOR*, pages 58–62. Springer–Verlag, 1999.
5. T. Fujito and H. Nagamochi. A 2-approximation algorithm for the minimum weight edge dominating set problem. *Discrete Appl. Math.*, 118:199–207, 2002.
6. M. R. Garey and D. S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness.* Freeman and Company, 1979.
7. E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *Siam Journal on Computing*, 31:1608–1623, 2002.
8. J.D. Horton and K. Kilakos. Minimum edge dominating sets. *SIAM J. Discrete Math.*, 6:375–387, 1993.
9. T. Imamura and K. Iwama. Approximating vertex cover on dense graphs. In *Proc. of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 582–589, 2005.

10. M. Karpinski and A. Zelikovsky. Approximating dense cases of covering problems. In P. Pardalos and D. Du, editors, *Proc. of the DIMACS Workshop on Network Design: Connectivity and Facilites Location*, volume 40 of *DIMACS series in Disc. Math. and Theor. Comp. Sci.*, pages 169–178, 1997.
11. H. Mélot. Facets Defining Inequalities among Graph Invariants: the system GraPHedron. Submitted, 2005.
12. A. Srinivasan, K. Madhukar, P. Navagamsi, C. Pandu Rangan, and M.-S. Chang. Edge domination on bipartite permutation graphs and cotriangulated graphs. *Inf.Proc. Letters*, 56:165–171, 1995.
13. M. Yannakakis and F. Gavril. Edge dominating sets in graphs. *SIAM J. Appl. Math.*, 38(3):364–372, 1980.

# A Randomized Algorithm for Online Unit Clustering*

Timothy M. Chan and Hamid Zarrabi-Zadeh

School of Computer Science, University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1
{tmchan,hzarrabi}@uwaterloo.ca

**Abstract.** In this paper, we consider the online version of the following problem: partition a set of input points into subsets, each enclosable by a unit ball, so as to minimize the number of subsets used. In the one-dimensional case, we show that surprisingly the naïve upper bound of 2 on the competitive ratio can be beaten: we present a new randomized 15/8-competitive online algorithm. We also provide some lower bounds and an extension to higher dimensions.

## 1 Introduction

Clustering problems—dividing a set of points into groups to optimize various objective functions—are fundamental and arise in a wide variety of applications such as information retrieval, data mining, and facility location. We mention two of the most basic and popular versions of clustering:

**Problem 1 ($k$-Center).** *Given a set of $n$ points and a parameter $k$, cover the set by $k$ congruent balls, so as to minimize the radius of the balls.*

**Problem 2 (Unit Covering).** *Given a set of $n$ points, cover the set by balls of unit radius, so as to minimize the number of balls used.*

Both problems are NP-hard in the Euclidean plane [10,19]. In fact, it is NP-hard to approximate the two-dimensional $k$-center problem to within a factor smaller than 2 [9]. Factor-2 algorithms are known for the $k$-center problem [9,11] in any dimension, while polynomial-time approximation schemes are known for the unit covering problem [14] in fixed dimensions.

Recently, many researchers have considered clustering problems in more practical settings, for example, in the online and data stream models [4,5,12], where the input is given as a sequence of points over time. In the online model, the solution must be constructed as points arrive and decisions made cannot be subsequently revoked; for example, in the unit covering problem, after a ball is opened to cover an incoming point, the ball cannot be removed later. In the related streaming model, the main concern is the amount of working space; as

* Work of the first author has been supported in part by NSERC.

T. Erlebach and C. Kaklamanis (Eds.): WAOA 2006, LNCS 4368, pp. 121–131, 2006.

points arrive, we must decide which point should be kept in memory. We focus on the online setting in this paper.

The online version of the unit covering problem is one of the problems addressed in the paper by Charikar et al. [4]. They have given an upper bound of $O(2^d d \log d)$ and a lower bound of $\Omega(\frac{\log d}{\log \log \log d})$ on the competitive ratio of deterministic online algorithms in $d$ dimensions; for $d = 1$ and 2, the lower bounds are 2 and 4 respectively.

In this paper, we address the online version of the following variant:

**Problem 3 (Unit Clustering).** *Given a set of $n$ points, partition the set into clusters (subsets), each of radius at most one, so as to minimize the number of clusters used. Here, the* radius *of a cluster refers to the radius of its smallest enclosing ball.*

At first glance, Problem 3 might look eerily similar to Problem 2; in fact, in the usual offline setting, they are identical. However, in the on-line setting, there is one important difference: as a point $p$ arrives, the unit clustering problem only requires us to decide on the choice of the cluster containing $p$, not the ball covering the cluster; the point cannot subsequently be reassigned to another cluster, but the position of the ball may be shifted.

We show that it is possible to get better results for Problem 3 than Problem 2. Interestingly we show that even in one dimension, the unit clustering problem admits a nontrivial algorithm with competitive ratio better than 2, albeit by using randomization. In contrast, such a result is not possible for unit covering. To be precise, we present an online algorithm for one-dimensional unit clustering that achieves expected competitive ratio 15/8 against oblivious adversaries. Our algorithm is not complicated but does require a combination of ideas and a careful case analysis. We contrast the result with a lower bound of 4/3 and also extend our algorithm for the problem in higher dimensions under the $L_\infty$ metric.

We believe that the one-dimensional unit clustering problem itself is theoretically appealing because of its utter simplicity and its connection to well-known problems. For example, in the exact offline setting, one-dimensional unit clustering/covering is known to be equivalent to the dual problem of finding a largest subset of disjoint intervals among a given set of unit intervals—i.e., finding maximum independent sets in unit interval graphs. Higher-dimensional generalizations of this dual independent set problem have been explored in the map labeling and computational geometry literature [2,3,8], and online algorithms for various problems about geometric intersection graphs have been considered (such as [18]). The one-dimensional independent set problem can also be viewed as a simple scheduling problem (dubbed "activity selection" by Cormen et al. [6]), and various online algorithms about intervals and interval graphs (such as [1,7,16,17]) have been addressed in the literature on scheduling and resource allocation. In the online setting, one-dimensional unit clustering is equivalent to clique partitioning in unit interval graphs, and thus, equivalent to coloring in unit co-interval graphs. It is known that general co-interval graphs can be colored with competitive ratio at most 2 [13], and that, no online deterministic

algorithm can beat this 2 bound [15]. To the best of our knowledge, however, online coloring of unit co-interval graphs has not been studied before.

## 2 Naïve Algorithms

In this section, we begin our study of the unit clustering problem in one dimension by pointing out the deficiencies of some natural strategies.

Recall that the goal is to assign points to clusters so that each cluster has length at most 1, where the *length* of a cluster refers to the length of its smallest enclosing interval. (Note that we have switched to using lengths instead of radii in one dimension; all intervals are closed.) We say that a point *lies* in a cluster if inserting it to the cluster would not increase the length of the cluster. We say that a point *fits* in a cluster if inserting it to the cluster would not cause the length to exceed 1. The following are three simple online algorithms, all easily provable to have competitive ratio at most 2:

**Algorithm 1** (CENTERED)**.** *For each new point $p$, if it is covered by an existing interval, put $p$ in the corresponding cluster, else open a new cluster for the unit interval centered at $p$.*

**Algorithm 2** (GRID)**.** *Build a uniform unit grid on the line (where cells are intervals of the form $[i, i+1)$). For each new point $p$, if the grid cell containing $p$ is nonempty, put $p$ in the corresponding cluster, else open a new cluster for the grid cell.*

**Algorithm 3** (GREEDY)**.** *For each new point $p$, if $p$ fits in some existing cluster, put $p$ in such a cluster, else open a new cluster for $p$.*

The first two algorithms actually solve the stronger unit covering problem (Problem 2). No such algorithms can break the 2 bound, as we can easily prove:

**Theorem 1.** *There is a lower bound of 2 on the competitive ratio of any randomized (and deterministic) algorithm for the online unit covering problem in one dimension.*

*Proof.* To show the lower bound for randomized algorithms, we use Yao's technique and provide a probability distribution on the input sequences such that the resulting expected competitive ratio for any deterministic online algorithm is at least 2. The adversary provides a sequence of 3 points at position 1, $x$, and $1 + x$, where $x$ is uniformly distributed in $[0, 1]$. The probability that a deterministic algorithm produces the optimal solution (of size 1 instead of 2 or more) is 0. Thus, the expected value of the competitive ratio is at least 2. □

The 2 bound on the competitive ratio is also tight for Algorithm 3: just consider the sequence $\langle \frac{1}{2}, \frac{3}{2}, \ldots, 2k - \frac{1}{2} \rangle$ followed by $\langle 0, 2, \ldots, 2k \rangle$ (where the greedy algorithm uses $2k+1$ clusters and the optimal solution needs only $k+1$ clusters). No random combination of Algorithms 1–3 can lead to a better competitive ratio, as we can easily see by the same bad example. New ideas are needed to beat 2.

## 3 The New Algorithm

In this section, we present a new randomized algorithm for the online unit clustering problem. While the competitive ratio of this algorithm is not necessarily less than 2, the algorithm is carefully designed so that when combined with Algorithm 2 we get a competitive ratio strictly less than 2.

Our algorithm builds upon the simple grid strategy (Algorithm 2). To guard against a bad example like $\langle \frac{1}{2}, \frac{3}{2}, \ldots \rangle$, the idea is to allow two points in different grid cells to be put in a common cluster "occasionally" (as controlled by randomization). Doing so might actually hurt, not help, in many cases, but fortunately we can still show that there is a net benefit (in expectation), at least in the most critical case.

To implement this idea, we form *windows* each consisting of two grid cells and permit clusters crossing the two cells within a window but try to "discourage" clusters crossing two windows. The details of the algorithm are delicate and are described below. Note that only one random bit is used at the beginning.

**Algorithm 4 (**RandWindow**).** *Group each two consecutive grid cells into a* window *of the form* $[2i, 2i+2)$*. With probability* $1/2$*, shift all windows one unit to the right. For each new point* $p$*, find the window* $w$ *and the grid cell* $c$ *containing* $p$*, and do the following:*

1: **if** $w$ is empty **then** open a new cluster for $p$
2: **else if** $p$ lies in a cluster **then** put $p$ in that cluster
3: **else if** $p$ fits in a cluster entirely inside $c$ **then** put $p$ in that cluster
4: **else if** $p$ fits in a cluster intersecting $w$ **then** put $p$ in that cluster
5: **else if** $p$ fits in a cluster entirely inside a neighboring window $w'$ and
6: $w'$ intersects $> 1$ clusters **then** put $p$ in that cluster
7: **else** open a new cluster for $p$

To summarize: the algorithm is greedy-like and opens a new cluster only if no existing cluster fits. The main exception is when the new point is the first point in a window (line 1); another exception arises from the (seemly mysterious) condition in line 6. When more than one cluster fits, the preference is towards clusters entirely inside a grid cell, and against clusters from neighboring windows. These exceptional cases and preference rules are vital to the analysis.

## 4 Analysis

For a grid cell (or a group of cells) $x$, the *cost* of $x$ denoted by $\mu(x)$ is defined to be the number of clusters fully contained in $x$ plus half the number of clusters crossing the boundaries of $x$, in the solution produced by our algorithm. Observe that $\mu$ is additive, i.e., for two adjacent groups of cells $x$ and $y$, $\mu(x \cup y) = \mu(x) + \mu(y)$. This definition of cost will be useful for accounting purposes.

To prepare for the analysis, we first make several observations concerning the behavior of the RandWindow algorithm. In the following, we refer to a cluster as a *crossing cluster* if it intersects two adjacent grid cells, or as a *whole cluster* if it is contained completely in a grid cell.

**Observation 1**

(i) *The enclosing intervals of the clusters are disjoint.*
(ii) *No grid cell contains two whole clusters.*
(iii) *If a grid cell $c$ intersects a crossing cluster $u_1$ and a whole cluster $u_2$, then $u_2$ must be opened after $u_1$ has been opened, and after $u_1$ has become a crossing cluster.*

*Proof.* (i) holds because of line 2. (ii) holds because line 3 precedes line 7.

For (iii), let $p_1$ be the first point of $u_1$ in $c$ and $p_1'$ be the first point of $u_1$ in a cell adjacent to $c$. Let $p_2$ be the first point of $u_2$. Among these three points, $p_1$ cannot be the last to arrive: otherwise, $p_1$ would be assigned to the whole cluster $u_2$ instead of $u_1$, because line 3 precedes lines 4–7. Furthermore, $p_1'$ cannot be the last to arrive: otherwise, $p_1$ would be assigned to $u_2$ instead, again because line 3 precedes lines 4–7. So, $p_2$ must be the last to arrive. □

For example, according to Observation 1(ii), every grid cell $c$ must have $\mu(c) \leq 1 + \frac{1}{2} + \frac{1}{2} = 2$.

Let $\sigma$ be the input sequence and $\texttt{opt}(\sigma)$ be an optimal covering of $\sigma$ by unit intervals, with the property that the intervals are disjoint. (This property is satisfied by some optimal solution, simply by repeatedly shifting the intervals to the right.) We partition the grid cells into blocks, where each *block* is a maximal set of consecutive grid cells interconnected by the intervals from $\texttt{opt}(\sigma)$ (see Fig. 1). Our approach is to analyze the cost of the solution produced by our algorithm within each block separately.

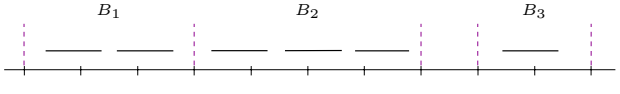



**Fig. 1.** Three blocks of sizes 2, 3, and 1

A block of size $k \geq 2$ contains exactly $k-1$ intervals from $\texttt{opt}(\sigma)$. Define $\rho(k)$ to be the competitive ratio of the RandWindow algorithm within a block of size $k$, i.e., $\rho(k)$ upper-bounds the expected value of $\mu(B)/(k-1)$ over all blocks $B$ of size $k$. The required case analysis is delicate and is described in detail below. The main case to watch out for is $k = 2$: any bound for $\rho(2)$ strictly smaller than 2 will lead to a competitive ratio strictly smaller than 2 for the final algorithm (as we will see in Section 5), although bounds for $\rho(3), \rho(4), \ldots$ will affect the final constant.

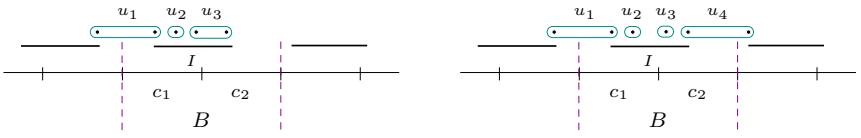



**Fig. 2.** Impossibility of Subcase 1.1 (left) and Subsubcase 1.3.2 (right)

**Theorem 2.** *$\rho(2) = 7/4$, $\rho(3) = 9/4$, $\rho(4) \le 7/3$, and $\rho(k) \le 2k/(k-1)$ for all $k \ge 5$.*

*Proof.* We first analyze $\rho(2)$. Consider a block $B$ of size 2, consisting of cells $c_1$ and $c_2$ from left to right. Let $I$ be the single unit interval in $B$ in $\mathtt{opt}(\sigma)$. There are two possibilities:

- Lucky Case: $B$ falls completely in one window $w$. After a cluster $u$ has been opened for the new point (by line 1), all subsequent points in $I$ are put in the same cluster $u$ (by lines 3 and 4). Note that the condition put in line 6 prevents points from the neighboring windows to join $u$ and make crossing clusters. So, $u$ is the only cluster in $B$, and hence, $\mu(B) = 1$.
- Unlucky Case: $B$ is split between two neighboring windows. We first rule out some subcases:
  - Subcase 1.1: $\mu(c_1) = 2$. Here, $c_1$ intersects three clusters $\langle u_1, u_2, u_3\rangle$ (from left to right), where $u_1$ and $u_3$ are crossing clusters and $u_2$ is a whole cluster (see Fig. 2, left). By Observation 1(iii), $u_2$ is opened after $u_3$ has become a crossing cluster, but then the points of $u_2$ would be assigned to $u_3$ instead (because line 4 precedes line 7 and $u_2 \cup u_3 \subset I$ has length at most 1): a contradiction.
  - Subcase 1.2: $\mu(c_2) = 2$. Similarly impossible.
  - Subcase 1.3: $\mu(c_1) = \mu(c_2) = 3/2$. We have only two scenarios:
    - Subsubcase 1.3.1: $B$ intersects three clusters $\langle u_1, u_2, u_3\rangle$, where $u_2$ is a crossing cluster, and $u_1$ and $u_3$ are whole clusters. By Observation 1(iii), $u_1$ is opened after $u_2$ has become a crossing cluster, but then the points of $u_1$ would be assigned to $u_2$ instead (because of line 4 and $u_1 \cup u_2 \subset I$): a contradiction.
    - Subsubcase 1.3.2: $B$ intersects four clusters $\langle u_1, u_2, u_3, u_4\rangle$, where $u_1$ and $u_4$ are crossing clusters and $u_2$ and $u_3$ are whole clusters (see Fig. 2, right). W.l.o.g., say $u_2$ is opened after $u_3$. By Observation 1(iii), $u_2$ is the last to be opened after $u_1, u_3, u_4$, but then $u_2$ would not be opened as points in $u_2$ may be assigned to $u_3$ (because lines 5–6 precedes line 7, $u_2 \cup u_3 \subset I$, and $c_2$ intersects more than one cluster): a contradiction.

  In all remaining subcases, $\mu(B) = \mu(c_1) + \mu(c_2) \le \frac{3}{2} + 1 = \frac{5}{2}$.

Since the lucky case occurs with probability exactly 1/2, we conclude that $\rho(2) \le \frac{1}{2}(1) + \frac{1}{2}(\frac{5}{2}) = \frac{7}{4}$. (This bound is tight.)

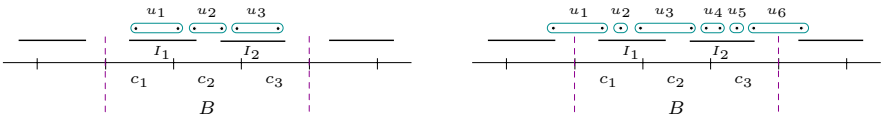



**Fig. 3.** Impossibility of Cases 2.1 (left) and 2.2 (right)

Next, we analyze $\rho(3)$. Consider a block $B$ of size 3, consisting of cells $c_1, c_2, c_3$ from left to right. (It will not matter below whether $c_1$ and $c_2$ fall in the same window, or $c_2$ and $c_3$ instead.) Let $I_1, I_2$ be the two unit intervals in $B$ in $\texttt{opt}(\sigma)$ from left to right.

- Case 2.1: $\mu(c_2) = 2$. Here, $c_2$ intersects three clusters $\langle u_1, u_2, u_3\rangle$ (from left to right), where $u_1$ and $u_3$ are crossing clusters and $u_2$ is a whole cluster (see Fig. 3, left). By Observation 1(iii), $u_2$ is opened after $u_1$ and $u_3$ have become crossing clusters, but then the points of $u_2$ would be assigned to $u_1$ or $u_3$ instead (because of line 4 and $u_1 \cup u_2 \cup u_3 \subset I_1 \cup I_2$): a contradiction.
- Case 2.2: $\mu(c_1) = \mu(c_3) = 2$. Here, $c_1$ intersects three clusters $\langle u_1, u_2, u_3\rangle$ and $c_3$ intersects three clusters $\langle u_4, u_5, u_6\rangle$ (from left to right), where $u_1$, $u_3$, $u_4$, $u_6$ are crossing clusters and $u_2, u_5$ are whole clusters (see Fig. 3, right). Then $u_3$ cannot be entirely contained in $I_1$: otherwise, by Observation 1(iii), $u_2$ is opened after $u_1$ and $u_3$ have become crossing clusters, but then the points of $u_2$ would be assigned to $u_3$ instead. Similarly, $u_4$ cannot be entirely contained in $I_2$. However, this implies that the enclosing intervals of $u_3$ and $u_4$ overlap: a contradiction.
- Case 2.3: $\mu(c_1) = 2$ and $\mu(c_2) = \mu(c_3) = 3/2$. Here, $B$ intersects six clusters $\langle u_1, \ldots, u_6\rangle$ (from left to right), where $u_1, u_3, u_6$ are crossing clusters and $u_2, u_4, u_5$ are whole clusters. As in Case 2.2, $u_3$ cannot be entirely contained in $I_1$. This implies that $u_4 \cup u_5 \subset I_2$. We now proceed as in Subcase 1.3.2. Say $u_4$ is opened after $u_5$ (the other scenario is symmetric). By Observation 1(iii), $u_4$ is the last to be opened after $u_3, u_5, u_6$, but then $u_4$ would not be opened as points in $u_4$ may be assigned to $u_5$: a contradiction.
- Case 2.4: $\mu(c_1) = \mu(c_2) = 3/2$ and $\mu(c_3) = 2$. Similarly impossible.

In all remaining subcases, $\mu(B) = \mu(c_1)+\mu(c_2)+\mu(c_3)$ is at most $2+\frac{3}{2}+1 = \frac{9}{2}$ or $\frac{3}{2}+\frac{3}{2}+\frac{3}{2} = \frac{9}{2}$. We conclude that $\rho(3) \leq 9/4$. (This bound is tight.)

Now, we analyze $\rho(4)$. Consider a block $B$ of size 4, consisting of cells $c_1, \ldots, c_4$ from left to right. Let $I_1, I_2, I_3$ be the three unit intervals in $B$ in $\texttt{opt}(\sigma)$ from left to right.

- Case 3.1: $\mu(c_1) = \mu(c_3) = 2$. Here, $c_1$ intersects three clusters $\langle u_1, u_2, u_3\rangle$ and $c_3$ intersects three clusters $\langle u_4, u_5, u_6\rangle$ (from left to right), where $u_1$, $u_3$, $u_4$, $u_6$ are crossing clusters and $u_2, u_5$ are whole clusters. As in Case 2.2, $u_3$ cannot be entirely contained in $I_1$. Thus, $u_4 \cup u_5 \cup u_6 \subset I_2 \cup I_3$. We now proceed as in Case 2.1. By Observation 1(iii), $u_5$ is opened after $u_4$ and $u_6$

have become crossing clusters, but then the points of $u_5$ would be assigned to $u_4$ or $u_6$ instead: a contradiction.
- Case 3.2: $\mu(c_2) = \mu(c_4) = 2$. Similarly impossible.

In all remaining subcases, $\mu(B) = (\mu(c_1) + \mu(c_3)) + (\mu(c_2) + \mu(c_4)) \leq (2 + \frac{3}{2}) + (2 + \frac{3}{2}) \leq 7$. We conclude that $\rho(4) \leq 7/3$.

For $k \geq 5$, we use a rather loose upper bound. Consider a block $B$ of size $k$. As each cell $c$ has $\mu(c) \leq 2$, we have $\mu(B) \leq 2k$, and hence $\rho(k) \leq 2k/(k-1)$. □

## 5 The Combined Algorithm

We can now combine the RandWindow algorithm (Algorithm 4) with the Grid algorithm (Algorithm 2) to obtain a randomized online algorithm with competitive ratio strictly less than 2. Note that only two random bits in total are used at the beginning.

**Algorithm 5 (Combo).** *With probability* 1/2*, run* RandWindow*, else run* Grid.

**Theorem 3.** Combo *is* 15/8*-competitive (against oblivious adversaries).*

*Proof.* The Grid algorithm uses exactly $k$ clusters on a block of size $k$. Therefore, the competitive ratio of this algorithm within a block of size $k$ is $k/(k-1)$.

The following table shows the competitive ratio of the RandWindow, Grid, and Combo algorithms, for all possible block sizes.

**Table 1.** The competitive ratio of the algorithms within a block

| Block Size | 2 | 3 | 4 | $k \geq 5$ |
|---|---|---|---|---|
| Grid | 2 | 3/2 | 4/3 | $k/(k-1)$ |
| RandWindow | 7/4 | 9/4 | $\leq 7/3$ | $\leq 2k/(k-1)$ |
| Combo | 15/8 | 15/8 | $\leq 11/6$ | $\leq 3/2 \cdot k/(k-1)$ |

As we can see, the competitive ratio of Combo within a block is always at most 15/8. By summing over all blocks and exploiting the additivity of our cost function $\mu$, we see that expected total cost of the solution produced by Combo is at most 15/8 times the size of $\texttt{opt}(\sigma)$ for every input sequence $\sigma$. □

We complement the above result with a quick lower bound argument:

**Theorem 4.** *There is a lower bound of* 4/3 *on the competitive ratio of any randomized algorithm for the online unit clustering problem in one dimension (against oblivious adversaries).*

*Proof.* We use Yao's technique. Consider two point sequences $P_1 = \langle 1, 2, \frac{1}{2}, \frac{5}{2} \rangle$ and $P_2 = \langle 1, 2, \frac{3}{2}, \frac{3}{2} \rangle$. With probability 2/3 the adversary provides $P_1$, and with probability 1/3 it provides $P_2$. Consider a deterministic algorithm $\mathcal{A}$. Regardless of which point sequence is selected by the adversary, the first two points provided to $\mathcal{A}$ are the same. If $\mathcal{A}$ clusters the first two points into one cluster, then it uses 3 clusters for $P_1$ and 1 cluster for $P_2$, giving the expected competitive ratio of $\frac{2}{3}(\frac{3}{2}) + \frac{1}{3}(1) = \frac{4}{3}$. If $\mathcal{A}$ clusters the first two points into two distinct clusters, then no more clusters are needed to cover the other two points of $P_1$ and $P_2$. Thus, the expected competitive ratio of $\mathcal{A}$ in this case is $\frac{2}{3} \cdot (1) + \frac{1}{3} \cdot (2) = \frac{4}{3}$ as well. □

## 6 Beyond One Dimension

In the two-dimensional $L_\infty$-metric case, we want to partition the given point set into subsets, each of $L_\infty$-diameter at most 1 (i.e., each enclosable by a unit square), so as to minimize the number of subsets used. (See Fig. 4.)

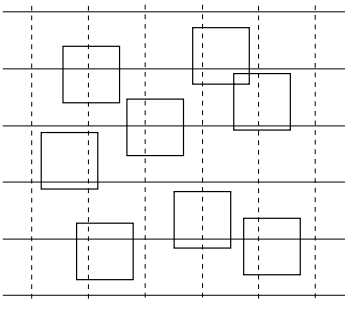


**Fig. 4.** Unit clustering in the $L_\infty$ plane

All the naïve algorithms mentioned in Section 2, when extended to two dimensions, provide 4-competitive solutions to the optimal solution. Theorem 1 can be generalized to a deterministic lower bound of 4 on the competitive ratio for the unit covering problem. We show how to extend Theorem 3 to obtain a competitive ratio strictly less than 4 for unit clustering.

**Theorem 5.** *There is a* 15/4*-competitive algorithm for the online unit clustering problem in the* $L_\infty$ *plane.*

*Proof.* Our online algorithm is simple: just use COMBO to find a unit clustering $C_i$ for the points inside each horizontal strip $i \le y < i + 1$. (Computing each $C_i$ is indeed a one-dimensional problem.)

Let $\sigma$ be the input sequence. We denote by $\sigma_i$ the set of points from $\sigma$ that lie in the strip $i \le y < i + 1$. Let $Z_i$ be an optimal unit covering for $\sigma_i$. Let $O$ be an optimal unit covering for $\sigma$, and $O_i$ be the set of unit squares in $O$ that intersect

the grid line $y = i$. Since all squares in $O_i$ lie in the strip $i-1 \leq y < i+1$, we have $|Z_i| \leq |O_{i-1}| + |O_i|$. Therefore $\sum_i |Z_i| \leq 2|O|$, so $\sum_i |C_i| \leq \frac{15}{8} \sum_i |Z_i| \leq \frac{15}{4}|O|$. □

The above theorem can easily be extended to dimension $d > 2$, with ratio $2^d \cdot 15/16$.

## 7 Closing Remarks

We have shown that determining the best competitive ratio for the online unit clustering problem is nontrivial even in the simplest one-dimensional case. The obvious open problem is to close the gap between the 15/8 upper bound and 4/3 lower bound. An intriguing possibility that we haven't ruled out is whether a nontrivial result can be obtained without randomization at all. There is an obvious 3/2 deterministic lower bound, but we do not see any simple argument that achieves a lower bound of 2.

We wonder if ideas that are more "geometric" may lead to still better results than Theorem 5. Our work certainly raises countless questions concerning the best competitive ratio in higher-dimensional cases, for other metrics besides $L_\infty$, and for other geometric measures of cluster sizes besides radius or diameter.

## References

1. U. Adamy and T. Erlebach. Online coloring of intervals with bandwidth. In *Proc. 1st Workshop Approx. Online Algorithms*, volume 2909 of *Lecture Notes Comput. Sci.*, pages 1–12, 2003.
2. P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Comput. Geom. Theory Appl.*, 11:209–218, 1998.
3. T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46:178–189, 2003.
4. M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004.
5. M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proc. 35th ACM Sympos. Theory Comput.*, pages 30–39, 2003.
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.
7. L. Epstein and M. Levy. Online interval coloring and variants. In *Proc. 32nd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 3580 of *Lecture Notes Comput. Sci.*, pages 602–613, 2005.
8. T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.*, 34:1302–1323, 2005.
9. T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th ACM Sympos. Theory Comput.*, pages 434–444, 1988.
10. R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.

11. T. Gonzalez. Covering a set of points in multidimensional space. *Inform. Process. Lett.*, 40:181–188, 1991.
12. S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proc. 41st IEEE Sympos. Found. Comput. Sci.*, pages 359–366, 2000.
13. A. Gyárfás and J. Lehel. On-line and First-Fit colorings of graphs. *J. Graph Theory*, 12:217–227, 1988.
14. D. S. Hochbaum and W. Maas. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32:130–136, 1985.
15. H. A. Kierstead and J. Qin. Coloring interval graphs with First-Fit. *SIAM J. Discrete Math.*, 8:47–57, 1995.
16. H. A. Kierstead and W. A. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.
17. R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proc. 5th Sympos. Discrete Algorithms*, pages 302–311, 1994.
18. M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.
19. N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM J. Comput.*, 13(1):182–196, 1984.

# Author Index