Applied Optimization

Josef Kallrath *Editor*

Algebraic Modeling Systems

Modeling and Solving Real World Optimization Problems

Springer

APPLIED OPTIMIZATION

For further volumes:
http://www.springer.com/series/5634

Applied Optimization

Series Editors:

Panos M. Pardalos
University of Florida, USA

Donald W. Hearn
University of Florida, USA

ALGEBRAIC MODELING SYSTEMS

Modeling and Solving Real World Optimization Problems

Edited by

Josef Kallrath

Springer

Prof. Dr. Josef Kallrath
Am Mahlstein 8
67273 Weisenheim
Germany

ISSN 1384-6485
ISBN 978-3-642-23591-7 e-ISBN 978-3-642-23592-4
DOI 10.1007/978-3-642-23592-4
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012931002

© Springer-Verlag Berlin Heidelberg 2012
This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.
The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To Julia, Diana and Albert

Foreword

The largely heterogeneous landscape of research activities in the domain of Algebraic Modeling Systems is reflected in the contributions that embody this book. In Part I, Chaps. 1–2 introduce the reader to the theme of the book. Chapter 1 provides an overview and focuses on the primary differences between algebraic modeling languages (e.g., GAMS, AMPL) and procedural languages (e.g., C, Fortran). Chapter 2 addresses how to translate an optimization problem from its original state to an algebraic modeling language. In Part II, Chaps. 3–7 contribute to different aspects of algebraic modeling systems. Chapter 3 introduces a framework for gathering, updating, and solving collections of models with related data that can be considered as scenarios. Chapter 4 provides a review of solution strategies for generalized disjunctive optimization models. Chapter 5 introduces the Mosel language, its solution interface for matrix-based solvers, and capabilities of handling multiple models, multiple problems within a model, problem decomposition, and distributed computing. Chapter 6 introduces the basic components of Constraint Programming Optimizer, CPO, and its application for scheduling problems. Chapter 7 presents an algebraic modeling language, ZIMPL, for mixed integer linear optimization problems and its applications in production planning. In Part III, Chaps. 8–12 address different aspects of modeling toward meeting the demands of real-world applications. Chapter 8 outlines the rationale for introducing algebraic modeling languages for optimization in semiconductor manufacturing. Chapter 9 describes the use of GNU R for data preparation in optimization models. Chapter 10 introduces a graphical tool which provides visualization of different objects, their relations, and their interactive functionality. Chapter 11 outlines the potential of algebraic modeling languages in the field of agricultural economics. Finally, Chap. 12 presents a wish list of excellent features and capabilities that, if introduced in existing algebraic modeling systems, could significantly enhance them. In sum, this collection of diverse and up-to-date contributions makes this book a very valuable addition to the library of all researchers and practitioners interested in mathematical modeling, optimization, and their applications.

Princeton University *Christodoulos A. Floudas*

Preface

This book *Algebraic Modeling Systems—Modeling and Solving Real World Optimization Problems*—deals with aspects of modeling and solving real-world optimization problems in a unique combination. It treats systematically the major algebraic modeling languages (AMLs) and algebraic modeling systems (AMSs) used to solve mathematical optimization problems. AMLs helped significantly to increase the usage of mathematical optimization in industry. Therefore, it is a logical consequence that the GOR (Gesellschaft für Operations Research) Working Group *Mathematical Optimization in Real Life* had a second meeting devoted to AMLs, which, after 7 years, followed the one held under the title *Modeling Languages in Mathematical Optimization* during April 23–25, 2003, in the German Physics Society Conference Building in Bad Honnef, Germany. Similar to the first meeting which resulted in the book *Modeling Languages in Mathematical Optimization*, this book is an offspring of the 86th Meeting of the GOR working group again held in Bad Honnef under the title *Modeling Languages in Mathematical Optimization—Overview, Opportunities & Challenges in Application Development*—during November 18–19, 2010. We note that some modeling languages have been sold from the original creators or developers and found a new haven. This time, the modeling language providers

AMPL Robert Fourer, Northwestern Univ.; David M. Gay, AMPL Optimization LLC., NJ,

CPLEX Studio Ferenc Katai, IBM ILOG SWG AIM, Valbonne, France,

GAMS Alexander Meeraus & Jan H. Jagla, GAMS Development Corp., Washington D.C.,

Mosel Susanne Heipcke & Oliver Bastert, FICO (previously, Dash Optimization),

gave deep insight into their motivations and conceptual software design features, highlighted their advantages, and also critically discussed their limits. These lectures were enhanced by presentations of practitioners sharing their experience with the audience, a talk about formal mathematical languages, and outlining the concept

of BoFiT, a Graphical Modeling Framework for Mixed Integer Programs. The participants benefited greatly from this symposium with its useful overview and orientation on today's modeling languages in optimization and their capabilities.

Roughly speaking, a modeling language serves the need to pass data and a mathematical model description to a solver in the same way that people, especially mathematicians, describe those problems to each other. Of course, in reality this is not done in exactly the same way, but the resemblance has to be close enough to spare the user any significant translation. As in this book we focus on modeling languages used in mathematical optimization, let us give an example from that discipline. When practitioners or researchers describe large-scale mixed integer linear programs (MILPs) to each other, they use summations and subscripting as well as domain specifications of the variables. To give a negative definition first: Probably one would not consider a language lacking these features to be a modeling language for large-scale MILP. This can be turned into a positive definition: A modeling language in mathematical optimization needs to support the expressions and symbols used in the mathematical optimization community. Therefore, it is natural that algebraic modeling languages support the concepts of data, variables, constraints, and objective functions. Those entities are connected not only by algebraic operations $(+, -, \cdot, /)$ but often also by nonlinear functions. Algebraic models are embedded in a larger class of differential-algebraic models including ordinary or partial differential equations. They may also be extended toward relationships appearing in constraint programming. Examples are membership relations or all-different constraints.

The earliest algebraic modeling languages appeared in the late 1970s and early 1980s. They were already very useful, supporting analysts to input their problems to solvers. In the first middle of the 1980s, when, for instance, `AMPL`, `GAMS`, `MPL`, and `mp-model` appeared, software developers were already trying to improve on previous designs, by taking advantage of faster computers and better computing environments.

Modeling languages supporting differential equations in addition to algebraic terms have their roots in the process industry and are covered by Kallrath [6]. Examples are `MINOPT`, `PCOMP`, and `gPROMS`.

While a more precise definition of modeling languages is given by Schichl [8], it is appropriate at this place to focus for a moment on the terms *modeling language* and *modeling systems*. Some people use these expressions synonymously, and this is also reflected in the acronyms `AMPL` and `GAMS`, where the former translates into *A Mathematical Programming Language* while the latter stands for *General Algebraic Modeling System*. In this book we rather keep the following meaning. In its purest sense, a modeling language in mathematical optimization is a means to give a declarative representation[1] of an optimization problem; `AMPL`, `GAMS`, and

[1] As the book, in the sense of this definition, focuses on algebraic declarative modeling languages, the reader should not be surprised to find not too much on mathematical modeling systems such as Mathematica, MathCad, MAPLE, or MATLAB. These systems are procedural tools.

mp-model are good examples. A modeling system is rather a complete support system to solve real-world problems. It usually contains a modeling language but offers many other features supporting the solution process, e.g., passing commands to the solver, analyzing infeasibilities, incorporating the solver's output back into the model or its next steps, or visualizing the branch&bound tree or the structure of the matrix; many more features could be listed. AIMMS, Mosel, MPL, and OPL Studio (now, CPLEX-Studio) are good examples of modeling systems. Of course, one might argue that the existing modeling languages are usually somewhat in between those extreme definitions. The early versions of AMPL were almost completely declarative, while GAMS from the beginning had also procedural features supporting, e.g., to input the output of one model into subsequent models.

As outlined by Kallrath (2011) in Chap. 1 of this book, AMLs have played and still play an important role in the mathematical optimization community and optimization used in industry. In the early 1980s, they found an initial market niche by enabling the user to formulate NLP problems; they supported automatic differentiation, i.e., they symbolically generated the first and second derivative information. AMLs and their broad distribution were triggered by the advent of personal computers (PCs). Dash Optimization with their solver XPRESS-OPTIMIZER and their modeling language mp-model provided a tool to PC users rather than mainframes. Thus, after a while, AMLs also became superior in implementing LP models and succeeded IBM's accepted industrial standard, MPS. Nowadays, the AMLs ensure the robustness, stability, and data checks needed in industrially stable software. Furthermore, AMLs accelerate the development and improvement of solvers ranging from Linear Programming to Mixed Integer Nonlinear Programming and even Global Optimization techniques. On one hand, users can easily switch from one solver to another one. On the other hand, the solver developers can count on a much larger market when their solver is embedded into an AML. Last but not least, the development of Microsoft Windows and improved hardware technology leads to graphical user interfaces such as Xpress-IVE for Mosel, GAMSIDE in GAMS, or systems such as AIMMS and MPL. This increases the efficiency of working with AMLs and contributes greatly to the fact that AMLs reduce the project time, make maintenance easier and extend the lifetime of optimization software.

This book is aimed at researchers of mathematical programming and operation research, scientists in various disciplines modeling and solving optimization problems, supply chain management consultants, and practitioners in the energy industry. It is beneficial to decision makers in the area of tool selection for optimization tasks as well as students and graduates in mathematics, physics, operations research, and businesses with interest in modeling and solving real optimization problems. Often application software has implemented an optimization model without an algebraic modeling language. The people responsible for maintaining or further developing such applications might be looking for improvements to put their software on a safer footing. They will definitely benefit from this book. Assuming some background in mathematics and optimization or at least a certain willingness to acquire the skills necessary to understand the described algorithms and models,

this book provides a sound overview on model formulation and solving as well as implementation of solvers in various modeling language packages. It demonstrates the strengths and characteristic features of such languages. May it provide a bridge for researchers, practitioners, and students into a new world of excitement: solving real-world optimization problems with the most advanced modeling systems.

Structure of this Book

This book benefits from contributions of experienced practitioners and developers of modeling languages and modeling system, respectively. The languages presented during the symposium as well as ZIMPL are covered in great detail in chapters of their own. We have structured the book in three parts:

- Introduction and Foundations
- Selected Algebraic Modeling Systems
- Aspects of Modeling and Solving Real World Problems

The book's first part contains introductory material. Chapter 1 starts with an introduction into AMLs, illuminates the meaning of the term model, and includes a brief overview on classes of optimization problems. The main entities in optimization—variables, constraints, and the objective function—are explained. The contrast between procedural and declarative languages is worked out in detail, followed by the importance of teaching and learning modeling. Chapter 2 discusses theoretical aspects and conceptual properties of semantic representation of mathematical specifications, an interesting aspect in the vicinity of declarative languages.

The second and main part of the book illuminates selected algebraic modeling systems. It covers CPLEX Studio (formerly, OPL Studio), GAMS, Mosel, and ZIMPL in great detail. Those chapters reflect the personal views and focus of the authors. It is left to the reader to compare the languages and see which one serves his personal needs best. The GAMS contribution focuses on the purposes of modeling languages with respect to different groups of models or customers, respectively, and describes a few special features of this modeling language. More on language elements related to constraint programming is found in the CPLEX Studio and Mosel chapters. A useful property in the modeling language Moselis its concept of *modularity*, which makes it possible to extend this language according to one's needs and to provide new functionality, in particular to deal with other types of problems and solvers. There are a few languages or systems not covered in this book because their developers or authors were not able to contribute to this book: Microsoft's solver suite and Excel plug-ins. Others have not seen an updated contribution; for those we refer the reader to the individual chapters in Kallrath [6]. In this group we find:

- AIMMS, which is quite different from all others due to its object-oriented design and provides agent-based simulation techniques [1].
- AMPL has a dominant position in universities, is often used for prototyping, and has many innovative ideas in its pipeline [3, 4].

- LPL with some neat features not found in any other language [5].
- LINGO with general global optimization algorithms which partition nonconvex models into a number of smaller convex models and then use a B&B manager to find the global optimal point over the entire variable domain [2].
- NOP-2, a special language by Schichl and Neumaier [9] for dealing with nonconvex nonlinear problems to be solved to global optimality.
- MPL [7] which has its strength on computer science and covers topics such as speed, scalability, data management, and deployment aspects of optimization.

The third part contains discussions of various aspects of modeling and solving real world problems among them reasons for selecting a procedural programming or a declarative modeling language (Chap. 8), data preparation (Chap. 9), visualization of data structures (Chap. 10), and a feature wish list and reflections about the current and possible future role of AMLs (Chaps. 11 and 12).

By showing the strengths and characteristic features of algebraic modeling languages as well as indicating trends, we hope to give novices and practitioners in mathematical optimization, operations research, supply chain management, energy industry, financial industry, and other areas of industry a useful overview. It may help decision makers to select the best modeling system satisfying their needs.

Weisenheim am Berg *Josef Kallrath*

References

1. Bisschop, J., Roelofs, M.: The modeling language AIMMS. In: Kallrath, J. (ed.) Modeling Languages in Mathematical Optimization, pp. 71–104. Kluwer Academic, Norwell (2004)
2. Cunningham, K., Schrage, L.: The LINGO algebraic modeling language. In: Kallrath, J. (ed.) Modeling Languages in Mathematical Optimization, pp. 159–171. Kluwer Academic, Norwell (2004)
3. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: a modeling language for mathematical programming. Duxbury Press, Brooks/Cole Publishing Company, Monterey (1993)
4. Fourer, R., Gay, D.M., Kernighan, B.W.: Design principles and new developments in the the AMPL modeling language. In: Kallrath, J. (ed.) Modeling languages in mathematical optimization, pp. 105–135. Kluwer Academic, Norwell (2004)
5. Hürlimann, T.: The LPL modeling language. In: Kallrath, J. (ed.) Modeling languages in mathematical optimization, pp. 173–183. Kluwer Academic, Norwell (2004)
6. Kallrath, J. (ed.): Modeling languages in mathematical optimization. Kluwer Academic, Norwell (2004)
7. Kristjansson, B., Lee, D.: The MPL modeling system. In: Kallrath, J. (ed.) Modeling languages in mathematical optimization, pp. 239–266. Kluwer Academic, Norwell (2004)
8. Schichl, H.: Theoretical concepts and design of modeling languages. In: Kallrath, J. (ed.) Modeling languages in mathematical optimization, pp. 45–62. Kluwer Academic, Norwell (2004)
9. Schichl, H., Neumaier, A.: The NOP-2 modeling language. In: Kallrath, J. (ed.) Modeling languages in mathematical optimization, pp. 279–291. Kluwer Academic, Norwell (2004)

Acknowledgements

Thanks is addressed at first to the contributing authors. It was a pleasure to work with such a group of distinguished individuals who formed the field and manifested their ideas into the modeling languages. The book benefited greatly from the fact that most of them contributed in time and used a clear LaTeX so that most of the editorial time was free to be spent on content and consistency. The book has been proofread by Dr. Steffen Rebennack and Timo Lohmann (Colorado School of Mines, Golden, CO), Dr. Susanne Heipcke (FICO, Marseille, France), and Dr. Peter Schodl (Vienna University, Vienna, Austria) gave valuable feedback and checked selected chapters. This book has been produced with Scientific Workplace kindly provided by Additive GmbH (Wiesbaden, www.additive-net.de). The publisher and his competent personnel (Christian Rauscher, Gabi Fischer and Deepak Ganesh) were very helpful while producing this book.

This book project has been sponsored by the major modeling language companies: FICO, GAMS GmbH, and IBM. Therefore, they deserve a strong expression of thanks and we give them some special covering in the acknowledgement section.

FICO (NYSE:FICO) is a leader in Decision Management, transforming business by making every decision count. FICO uses predictive analytics and optimization to help businesses automate, improve, and connect decisions across organizational silos and customer life cycles. FICO develops and markets the FICO Xpress Optimization Suite—with its unique `Mosel` modeling language (www.fico.com/xpress). Clients in 80 countries work with FICO to increase customer loyalty and profitability, cut fraud losses, manage credit risk, meet regulatory and competitive demands, and rapidly build market share. Most leading banks and credit card issuers rely on FICO solutions, as do insurers, retailers, health care organizations, and other companies. FICO works with more than 5,000 businesses worldwide, and its technology serves thousands more through FICO partnerships.

The General Algebraic Modeling System (`GAMS`) was the first high-level modeling system for mathematical programming problems. Thirty-five plus years of experience and development make `GAMS` a unique software package, ideally suited for building fully portable large-scale modeling applications. GAMS' commitment to backward compatibility and continuous updates of the software protect the

investment of their users. Given the strong will to bring new ideas from research facilities to practice, GAMS bridges the gap between academia and industry. GAMS is used in corporations and research laboratories in more than 120 countries around the world, as well as transnational organizations like EC, NATO, OECD, UN, WTO, etc. Both GAMS Development Corporation and GAMS Software GmbH are proud to serve a growing global GAMS user community. In recognition of their dedication to furthering the advance of OR, GAMS was awarded the INFORMS Computing Society Prize (1988) and the GOR company award (2010).

Operations Research, an engineering and management discipline dedicated to removing system inefficiencies, provides the decision support technologies to address them. IBM has been a major contributor to this field since the 1960s, developing and commercializing many of the statistical and mathematical techniques of Advanced Analytics. IBM ILOG Optimization helps companies quickly determine how to most effectively use limited resources, helping businesses to create the best possible plans, explore alternatives, understand trade-offs, and respond to changes in business environment.

Contents

Contributors

Wolfgang Britz Institute for Food and Resource Economics, University Bonn, Bonn, Germany

Michael R. Bussieck GAMS Software GmbH, Cologne, Germany

Ferenc Domes Fakultät für Mathematik, Universität Wien, Wien, Austria

Ulrich Dorndorf INFORM GmbH, Aachen, Germany

Stefan Droste INFORM GmbH, Aachen, Germany

Michael C. Ferris University of Wisconsin, Madison, WI, USA

Hermann Gold Infineon Technologies AG, Regensburg, Germany

Ignacio E. Grossmann Carnegie Mellon University, Pittsburgh, PA, USA

Axel Hecker Mathesis GmbH, Mannheim, Germany

Susanne Heipcke Xpress Team, FICO, FICO House, Starley Way, Birmingham, UK

Arnd vom Hofe Mathesis GmbH, Mannheim, Germany

Jan-H. Jagla GAMS Development Corp., Washington, DC, USA

Josef Kallrath BASF SE, Scientific Computing, Ludwigshafen, Germany Department of Astronomy, University of Florida, Gainesville, FL, USA

Ferenc Katai IBM, Gentilly Cedex, France

Thorsten Koch Zuse Institut Berlin, Berlin, Germany

Philippe Laborie IBM, Gentilly Cedex, France

Timo Lohmann Colorado School of Mines, Golden, CO, USA

Arnold Neumaier Fakultät für Mathematik, Universität Wien, Wien, Austria

Kevin Kofler Fakultät für Mathematik, Universität Wien, Wien, Austria

Alex Meeraus GAMS Development Corp., Washington, DC, USA

Jérôme Rogerie IBM, Gentilly Cedex, France

Juan P. Ruiz Carnegie Mellon University, Pittsburgh, PA, USA

Hermann Schichl Fakultät für Mathematik, Universität Wien, Wien, Austria

Peter Schodl Fakultät für Mathematik, Universität Wien, Wien, Austria

Paul Shaw IBM, Gentilly Cedex, France

Aldo Vecchietti Universidad Tecnológica Nacional, Santa Fe, Argentina

Petr Vilím IBM, Gentilly Cedex, France

Acronyms

Lists of abbreviations used in this book:

AML	Algebraic modeling language
AMS	Algebraic modeling systems
ATP	Available to promise
BARON	Branch and reduce optimization navigator
B&B	Branch and bound
B&C	Branch and cut
B&P	Branch and price
BM	Big-M
BONMIN	Basic open-source nonlinear mixed integer programming
CGE	Computable general equilibrium
CP	Constraint programming
CRAN	Comprehensive R archive network
CTP	Capable to promise
CTM	Capable to match
CV	Cross validation
DEA	Data envelopment analysis
DICOPT	Discrete and continuous optimizer
EMP	Extended mathematical programming
ERP	Enterprise resource planning
GAMS	General algebraic modeling system
GDP	Generalized disjunctive programming
GOR	Gesellschaft fr operations research
GUI	Graphical user interface
HR	Hull relaxation
IVE	Integrated visual environment
LBOA	Logic-based outer approximation
LogMIP	Logical mixed integer programming
LP	Linear programming
MCM	Multi-commodity model

MILP	Mixed integer linear programming
MINLP	Mixed integer nonlinear programming
MIP	Mixed integer programming
NLP	Nonlinear programming
PMP	Positive mathematical programming
PPM	Production process model
QP	Quadratic programming
SBB	Standard branch and bound
SDDP	Stochastic dual dynamic programming
SLP	Successive linear programming
SNP	Supply network planning
XAD	Xpress application developer

Part I
Introduction and Foundations

This part gives an introduction and overview on algebraic modeling languages and systems as well as some theoretical foundations.

Chapter 1 starts with an introduction into algebraic modeling languages, illuminates the meaning of the term model, and includes a brief overview on classes of optimization problems. The main entities in optimization—variables, constraints, and the objective function—are explained. The contrast between procedural and declarative languages is worked out in detail, followed by the importance of teaching and learning modeling. Chapter 2 discusses theoretical aspects and conceptual properties of semantic representation of mathematical specifications, an interesting aspect in the vicinity of declarative languages.

Chapter 1
Algebraic Modeling Languages: Introduction and Overview

Josef Kallrath

Abstract This chapter introduces the reader to algebraic modeling languages and their role in the mathematical optimization community. It focuses on the differences of procedural languages such as C, Fortran, MATLAB and others in comparison to the declarative algebraic modeling languages. This is more than a language or syntax issue but has deep roots in the concept of modeling and university teaching.

1.1 Introduction

A modeling language used to implement mathematical optimization models needs to support the expressions and symbols used in the mathematical optimization community. Therefore, it is natural that algebraic modeling languages (AMLs) support the concepts of data (what is given), variables (what we want to know), constraints (restrictions, bounds, etc.) and objective function (what we want to maximize or minimize). Those entities are not only connected by the algebraic operations (+,-,·) but also by nonlinear functional relationships. Algebraic modeling languages (AMLs)—the earliest, `GAMS`, `LINGO` and `mp-model`, appeared in the late 1970s and early 1980s—are declarative languages for implementing optimization problems. They keep the relations (equalities and inequalities) and restrictions among the variables, and connect data and the mathematical objects to a solver; they do not contain information on *how* to solve the optimization problem.

Since the early 1980s, AMLs have played and still play an important role in the mathematical optimization community and optimization used in industry. In the 1950s and 1960s, Assembler and Fortran coded LP models were mostly replaced

J. Kallrath (✉)
BASF SE, Scientific Computing, GVM/S-B009, D-67056 Ludwigshafen, Germany
e-mail: josef.kallrath@web.de

Department of Astronomy, University of Florida, Gainesville, FL 32611, USA
e-mail: kallrath@astro.ufl.edu

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_1, © Springer-Verlag Berlin Heidelberg 2012

by IBM's matrix generators MPS that established the standard of industrial model formulation; and models in these days were LP models many of them solved by IBM's LP solver MPSX. At that time there was no market for AMLs. But, there was no real support for NLP problems—and this was a niche for AMLs as they enabled the user to formulate NLP problems, and supported automatic differentiation, i.e., they symbolically generated the first and second derivative information. Another line of development was triggered by the advent of personal computers (PCs). Dash Optimization with their solver XPRESS-OPTIMIZER and their modeling language `mp-model` provided a tool to PC users rather than mainframes. Thus, after a while, AMLs also became superior in implementing LP models and succeeded MPS. Nowadays, academic research models (developed by scientists) are used to developing and testing solvers, or constructing efficient model reformulations. Domain expert models (developed by analysts) are used within consulting projects, or feasibility studies. And finally, AMLs often host the models for black box model users doing their operational planning. The AMLs ensure the robustness, stability, and data checks needed in industrially stable software. Furthermore, AMLs accelerate the development and improvement of solvers ranging from Linear Programming to Mixed Integer Nonlinear Programming and even Global Optimization techniques. If a user has an NLP problem implemented in an AML using a local solver to computes its local optimum, it is only a matter of minutes to switch to a global solver such as BARON or LINDOGLOBAL. Thus, there is a significantly reduced development risk for the user. But also the solver developers can count on a much larger market when their solver is embedded into an AML. The solver technology, in some sense, is now a commodity which allows the users to switch, for instance, from one MILP solver to another one, or play and collect experience with the free Coin-OR solvers. The implementation of polylithic modeling and solution approaches described in Kallrath [7] is possible without huge development efforts. And last but not least, the development of Microsoft Windows and improved hardware technology lead to graphical user interfaces such as Xpress-IVE for Mosel, GAMSIDE in `GAMS`, or systems such as `AIMMS` and `MPL`. This increases the efficiency of working with AMLs and contributes greatly to the fact that AMLs reduce the project time, make maintenance easier and increasing the lifetime of optimization software.

Nowadays, the most frequently used ones are in alphabetic order:

- `AIMMS`: Historically, `AIMMS` is a deviation of the `GAMS` development towards a more graphical user interface approach with an objected orientated background. The company, Paragon, who owns and develops `AIMMS`, won the Edelmann award in 2011 together with Midwest ISO and Alstom Grid.
- `AMPL`: Robert Fourer and David Gay are in charge and the owners of `AMPL`. It has a dominant position in universities, is often used for prototyping and has many innovative ideas in its pipeline.
- `Cplex-Studio`: Formerly, this was OPL-Studio developed by Pascal van Hentenryk and then part of ILOG; now it is owned by IBM.

- GAMS has its roots in the World Bank, where, in the late 1970s Alexander Meeraus and Jan Bisschop developed the first version before it became public in 1982.
- LINGO with general global optimization algorithms which partition nonconvex models into a number of smaller convex models and then use a B&B manager to find the global optimal point over the entire variable domain [3],
- LPL: Solely developed by Tony Hürliman (Fribourg University, Fribourg, Switzerland) it has some unique handy features.
- MPL: Developed by Bjarni Kristjansson in the 1980s, MPL [10] has its strenghts when it comes to scalability, memory management and other IT aspects, i.e., speed, scalability, data management and deployment aspects of optimization,
- Mosel is the successor of mp-model, which has been developed by Robert C. Daniel and Robert Ashford and was from 1983 to 2001 the language sold with Xpress-MP (originally, the LP solver LP-OPT, shortly after this enhanced by the MILP solver, MP-OPT) by Dash Optimization. Since 2001, Mosel (designed and developed by Yves Colombani) is the modeling environment for Xpress.
- ZIMPL is a language which has been developed by Thorsten Koch from the Konrad-Zuse Centrum in Berlin. Next to LPL, it is the only development from an academic institution.

The reader might forgive, if some language is missing. It is not that there are many new ones developed these days, but on this planet there simply is such a menagerie of modeling languages—and some of them I may have just not counted as AMLs.

1.2 Models and the Contrast Between Declarative and Procedural

As outlined in Kallrath and Maindl (Sect. 2.1, [8]) modeling is an important concept and methodology in various disciplines.The terms *modeling* or *model building* are derived from the word *model*. Its ethymological roots are the Latin word *modellus* (scale, [diminutiv of modus, measure]) and what was to be in the sixteenth century the new word *modello*. Nowadays, in the scientific context, the term is used to refer to a simplified, abstract or well structured partial aspect of the reality one is interested in. The idea itself and the associated concept is, however, much older. The classical geometry and especially Phythagoras around 600 B.C. distinguish between wheel and circle, between field and rectangle. Around 1,100 D.C. a wooden model of the later Speyer cathedral was produced; the model served to build the real cathedral. Astrolabs and celestial globes have been used as models to visualize the movement of the moon, planets and stars on the celestial sphere and to compute the times of rises and settings. Until the nineteenth century mechanical models were understood as pictures of the reality. Following the principals of classical mechanics the key idea was to reduce all phenomena to the movement of small particles. In physics and other mathematical sciences one talks about models if:

- One, for reasons of simplifications, restricts oneself to certain aspects of the problem (example: if we consider the movement of the planets, in a first approximation the planets are treated as point masses).
- One, for reasons of didactic presentation, develops a simplified picture for the more complicated reality (example: the planetary model is used to explain the situation inside the atoms).
- One uses the properties in one area to study the situation in an analogue problem.

A model is referred to as a mathematical model of a process or a problem if it contains the typical mathematical objects (variables, terms, relations). Thus, a (mathematical) model represents a real world problem in the language of mathematics using mathematical symbols, variables, equations, inequalities and other relations.

When building a model it is important to define and state precisely the purpose of the model. In science, we often encounter epistemological arguments. In engineering, a model might be used to construct some machines. In operations research and optimization, models are often used to support strategic or operative decisions. All models enable us to

- Learn and understand situations which do not allow easy access (very slow or fast processes, processes involving a very small or very large region).
- Avoid difficult, expensive or dangerous experiments; and analyze case studies and *what-if* scenarios.

A declarative model in operations research or optimization contains (algebraic) constraints to implicitly represent an optimization problem's feasible region, i.e., to restrict the variables. It does not say or know anything about how to solve the optimization problem. The declarative character is especially useful, as the feasible region is often a continuous subset of the n-dimensional space $\mathbb{R}^n$, which makes it impossible to list explicitly all feasible points. To understand this, see the following example: The inequality $x^2 + y^2 \leq R^2$ in connection with the bounds $-R \leq x \leq R$ and $0 \leq y \leq R$ expresses that the feasible region $\mathscr{R}$ is the upper half circle with radius R. The objective function $\max(R/3 - x)$ asks us to find the point within $\mathscr{R}$ that maximizes the function $R/3 - x$. In this case it is possible to compute the solution analytically:

$$(x, y)_* = \frac{1}{6} R \left(1 - \sqrt{17}, 3 + \frac{1}{3}\sqrt{17} \right). \tag{1.1}$$

AMLs and their declarative character are ideal to implement mixed integer programming (MIP) problems specified by the augmented vector $\mathbf{x}_{\oplus}^{\mathrm{T}} = \mathbf{x}^{\mathrm{T}} \oplus \mathbf{y}^{\mathrm{T}}$ established by vectors $\mathbf{x}^{\mathrm{T}} = (x_1, \ldots, x_{n_c})$ and $\mathbf{y}^{\mathrm{T}} = (y_1, \ldots, y_{n_d})$ of n_c continuous and n_d discrete variables, an objective function $f(\mathbf{x}, \mathbf{y})$, n_e equality constraints $\mathbf{h}(\mathbf{x}, \mathbf{y})$ and n_i inequality constraints $\mathbf{g}(\mathbf{x}, \mathbf{y})$. The problem

$$\min \left\{ f(\mathbf{x},\mathbf{y}) \;\middle|\; \begin{array}{l} \mathbf{h}(\mathbf{x},\mathbf{y}) = 0 \;\; \mathbf{h} : X \times U \to \mathbb{R}^{n_e} \;\; \mathbf{x} \in X \subseteq \mathbb{R}^{n_c} \\ \mathbf{g}(\mathbf{x},\mathbf{y}) \geq 0, \; \mathbf{g} : X \times U \to \mathbb{R}^{n_i}, \; \mathbf{y} \in U \subseteq \mathbb{Z}^{n_d} \end{array} \right\} \tag{1.2}$$

is called *Mixed Integer Nonlinear Programming* (MINLP) problem, if at least one of the functions f, $\mathbf{g}$ or $\mathbf{h}$ is nonlinear. The vector inequality, $\mathbf{g}(\mathbf{x},\mathbf{y}) \geq 0$, is to be read component-wise. Any vector $\mathbf{x}_{\oplus}^{\mathrm{T}}$ satisfying the constraints of (1.2) is called a *feasible point* of (1.2). Any feasible point, whose objective function value is less or equal than that of all other feasible points is called an *optimal solution*. Depending on the functions f, $\mathbf{g}$, and $\mathbf{h}$ in (1.2) we get the standard problems types

Acronym	*Type of optimization*	$f(\mathbf{x},\mathbf{y})$	$\mathbf{h}(\mathbf{x},\mathbf{y})$	$\mathbf{g}(\mathbf{x},\mathbf{y})$	n_d
LP	Linear Programming	$\mathbf{c}^{\mathrm{T}}\mathbf{x}$	$\mathsf{A}\mathbf{x} - \mathbf{b}$	$\mathbf{x}$	0
QP	Quadratic Programming	$\mathbf{x}^{\mathrm{T}} Q \mathbf{x} + \mathbf{c}^{\mathrm{T}}\mathbf{x}$	$\mathsf{A}\mathbf{x} - \mathbf{b}$	$\mathbf{x}$	0
NLP	Nonlinear Programming				0
MILP	Mixed Integer LP	$\mathbf{c}^{\mathrm{T}}\mathbf{x}_{\oplus}$	$\mathsf{A}\mathbf{x}_{\oplus} - \mathbf{b}$	$\mathbf{x}_{\oplus}$	≥ 1
MIQP	Mixed Integer QP	$\mathbf{x}_{\oplus}^{\mathrm{T}} \mathsf{Q} \mathbf{x}_{\oplus} + \mathbf{c}^{\mathrm{T}}\mathbf{x}_{\oplus}$	$\mathsf{A}\mathbf{x}_{\oplus} - \mathbf{b}$	$\mathbf{x}_{\oplus}$	≥ 1
MINLP	Mixed Integer NLP				≥ 1

with a matrix $\mathsf{A} \in \mathscr{M}(m \times n, \mathbb{R})$ of m rows and n columns, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$, and $n = n_c + n_d$. MIPs without continuous variables are integer programs (IPs). Beyond these types, there are other special mathematical programming types such as cone programming, mixed integer complementarity problems, mathematical programming with equlibrium constraints (MPEC).

Imperative *languages*, sometimes also called *procedural languages* or *algorithmic languages*, with the programming languages Algol, C, Fortran, and Pascal being the most prominent ones, represent a model of a problem almost identically to how the algorithm proceeds to solve this problem; cf. Hürlimann [5]. A computer program that computes the reflectance of light onto a car using Monte Carlo techniques embedded in the physics of radiation transfer and Fresnel's Laws, could therefore be seen as the implementation of a physical model for the reflection of light onto the coating of a car.

In the beginning, most algebraic modeling languages supported algebraic equalities and inequalities, and were purely declarative. When one builds a mathematical optimization model, and, especially, if one wants to use polylithic modeling and solution approaches [7] there is always some need for procedural extensions. How to deal with procedural aspects distinguishes the various AMLs and reflects their architecture. `MPL` and `AMPL` kept their pure declarative characters using optimax or scripts to cover the procedural needs. The orthogonal approach is seen in `GAMS`, `AIMMS` and `LINGO` with procedural commands, e.g., *loop*, *for*, *if-else*, *while* and others added to the declarative commands. `Cplex-Studio` and `Mosel` construct the models, i.e., the constraint system is procedurally generated.

1.3 Example Problem: Declarative Versus Procedural

Let us consider the following example based on the German soccer league. For each game won the team gets 3 points, draw the team gets 1 points, and lost the team gets 0 points. After G games a soccer team has collected P points. With w, d, and l denoting the number of games won, draw and lost, we have the relations

$$w + d + l = G \tag{1.3}$$

$$3w + d = P \tag{1.4}$$

$$w, d, l \in N_0 := \{0, 1, 2, \ldots\} \tag{1.5}$$

The tasks are to find:

1. All possible combinations of games won, draw and lost,
2. The combination with the largest numbers of games won, and
3. The combination with the smallest numbers of games won.

Besides the general solution we would like to solve a specific problem instance with $G = 14$ and $P = 28$. Let us indicate (in a humorous way) a few solution approaches how people from various disciplines would approach this problem:

1. A *Discrete Mathematician* would approach the problem by deriving a diophantic equation and parametrize the solutions by $k \in Z$ and $(w, d, l) = (w, d, l)(k)$. Eliminating d from (1.3) and (1.4), leads to the diophantic equation

   $$2w - l = P - G. \tag{1.6}$$

   For $G = 14$ and $P = 28$, (1.6) has the solution

   $$l = 2k, \quad w = 7 + k; \quad k \in \{0, 1, 2\} \tag{1.7}$$

   Due to the non-negativity constraint, we get the solutions

k	$w = 7 + k$	$d = 7 - 3k$	$l = 2k$
0	7	7	0
1	8	4	2
2	9	1	4

   (1.8)

   with 7–9 games won.
2. An IT student with no mathematics background would program a slow, brute force approach with 3 nested for loops in w, d, and l checking whether the combination gives G games and P points.
3. A mathematics student would apply a little bit of algebra and gets the additional relations

$$w = (P - G + l)/2 \tag{1.9}$$

$$d = G - w - l = \frac{1}{2}(3G - P - 3l) \tag{1.10}$$

and probably programs an efficient enumeration scheme depending only on l which eliminates most unreasonable combinations. For $G = 14$ and $P = 28$, only the values $l \in \{0, 2, 4\}$ lead to feasible combinations.

4. A user of algebraic modeling languages would probably formulate the integer programming model $\max w$ (or, $\min w$, resp.) subject to (1.3) and (1.4) and the bound $w \geq \lceil (P - G)/2 \rceil$ which is easily solved with any MILP solver. Solvers such as CPLEX, XPRESS-OPTIMIZER or BARON allow the user also to compute all integer feasible integer solutions which would be the appropriate way to find *all combinations* (in that case, the dummy objective function $x = 1$ with an auxiliary variable x would do).

For this little example, all approches presented work fine. But for larger values of P and G performance matters.

1.4 Modeling and Teaching: Conclusions

In Sect. 1.3 we tried to pass the idea that solution approaches depend on one's background. This background is defined by education, exposure to different techniques, and experience. In the engineering community, for instance, MATLAB is taught at many places, and therefore it is very natural for engineers in their post-university life to try to solve a problem using MATLAB. Mathematicians and physicists, usually, have a good training in Fortran or C to code their own algorithms to solve their problems often involving differential equations. There might be some chemical engineering, OR, or economic schools where students are exposured, for instance, to GAMS and develop some familiarity with the declaritive approach. But in many business schools, most probably, Excel is the tool of choice.

The problem with modeling and teaching is that even in optimization courses, the focus is less if at all on modeling but rather on how to solve problems. Modeling, i.e., the process of transforming a real world problem to a mathematical structure, is rarely systematically taught because it is not part of the curriculum—and this is usually so because those who design the course of study of, for instance, mathematics or bussiness, do not consider modeling as important. Mathematicians sometimes do not even see modeling as a part of mathematics.

Learning and teaching modeling is supported by a few very recommended books: Williams [11], Kallrath and Wilson [9], Bisschop [1], Kallrath [6], Castillo et al. [2] and Heipcke et al. [4]. I enthusiastically recommend to use AMLs in university courses as it helps to focus on mapping the real world problem to a mathematical model and not wasting too much time on implementation in procedural languages

or the details of tableau simplex iterations (modern solvers work anyway different from textbook descriptions of the Simplex algorithm).

Overall, each AML has its communities and ways to attract new modelers to use their language. The community spirit helps: For instance, agricultural people or chemical engineers recommend to their new students the language `GAMS` as this is what they use themselves. Industrial projects realized by Paragon or IBM often use the companies languages `AIMMS` and `CPLEX Studio`. `AMPL` is used frequently in universities because it has specials arrangements? Many students use `LINGO` as it is simple and affordable by them. Some people use several languages depending on their needs and suitability.

Acknowledgements It is a pleasure to thank Michael Bussieck (GAMS GmbH, Braunschweig, Germany), Susanne Heipcke (FICO, Marseille, France) and Steffen Rebennack (Coloroda School of Mines, Golden, CO) for their feedback and discussions on this chapter.

References

1. Bisschop, J.: AIMMS Optimization Modeling. Paragon Decision Technology B.V., DG Haarlem, The Netherlands (1999)
2. Castillo, E., Conejo, A.J., Garcia, P.P.R., Alguacil, N.: Building and Solving Mathematical Programming Models in Engineering and Science. John Wiley and Sons, Chichester (2002)
3. Cunningham, K., Schrage, L.: The LINGO Algebraic Modeling Language. In: J. Kallrath (ed.) Modeling Languages in Mathematical Optimization, pp. 159–171. Kluwer Academic Publishers, Norwell, MA, USA (2004)
4. Guéret, C., Heipcke, S., Prins, C., Sevaux, M.: Applications of Optimization with Xpress-MP. Dash Optimization, Blisworth, UK (2002). URL http://optimization.fico.com/product-information/, http://examples.xpress.fico.com/example.pl#mosel_book
5. Hürlimann, T.: The LPL Modeling Language. In: J. Kallrath (ed.) Modeling Languages in Mathematical Optimization, pp. 173–183. Kluwer Academic Publishers, Norwell, MA, USA (2004)
6. Kallrath, J.: Gemischt-Ganzzahlige Optimierung: Modellierung in der Praxis. Vieweg, Wiesbaden, Germany (2002)
7. Kallrath, J.: Polylithic Modeling and Solution Approaches Using Algebraic Modeling Systems. Optimization Letters **5**, 453–466 (2011). 10.1007/s11590-011-0320-4
8. Kallrath, J., Maindl, T.I.: Real Optimization with SAP-APO. Springer, Heidelberg, Germany (2006)
9. Kallrath, J., Wilson, J.M.: Business Optimisation Using Mathematical Programming. Macmillan, Houndmills, Basingstoke, UK (1997)
10. Kristjansson, B., Lee, D.: The MPL Modeling System. In: J. Kallrath (ed.) Modeling Languages in Mathematical Optimization, pp. 239–266. Kluwer Academic Publishers, Norwell, MA, USA (2004)
11. Williams, H.P.: Model Building in Mathematical Programming, 3rd edn. John Wiley and Sons, Chichester (1993)

Chapter 2
Towards a Self-Reflective, Context-Aware Semantic Representation of Mathematical Specifications

Peter Schodl, Arnold Neumaier, Kevin Kofler, Ferenc Domes, and Hermann Schichl

Abstract We discuss a framework for the representation and processing of mathematics developed within and for the MoSMath project. The MoSMath project aims to create a software system that is able to translate optimization problems from an almost natural language to the algebraic modeling language AMPL. As part of a greater vision (the FMathL project), this framework is designed both to serve the optimization-oriented MoSMath project, and to provide a basis for the much more general FMathL project.
We introduce the semantic memory, a data structure to represent semantic information, a type system to define and assign types to data, and the semantic virtual machine (SVM), a low level, Turing-complete programming system that processes data represented in the semantic memory.
Two features that set our approach apart from other frameworks are the possibility to reflect every major part of the system within the system itself, and the emphasis on the context-awareness of mathematics.
Arguments are given why this framework appears to be well suited for the representation and processing of arbitrary mathematics. It is discussed which mathematical content the framework is currently able to represent and interface.

2.1 The MoSMath Project

The project "a modeling system for mathematics" (MoSMath), currently carried out at the University of Vienna, aims to create a modeling system for the specification of models for the numerical work in optimization in a form that is natural for the working mathematician. The specified model is represented and processed

P. Schodl · A. Neumaier (✉) · K. Kofler · F. Domes · H. Schichl
Fakultät für Mathematik, Universität Wien, Nordbergstr. 15, A-1090 Wien, Austria
e-mail: peter.schodl@univie.ac.at; arnold.neumaier@univie.ac.at; kevin.kofler@chello.at; ferenc.domes@univie.ac.at; hermann.schichl@univie.ac.at

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_2, © Springer-Verlag Berlin Heidelberg 2012

inside a framework and can then be communicated to numerical solvers or other systems. While the input format is a controlled natural language like Naproche [6] and MathNat [9], but with a different target, it is designed to be as expressive and natural as currently feasible. This paper summarizes the work done in our group, with emphasis on the content of the PhD thesis of the first author.

The user benefits from this input format in multiple ways: The most obvious advantage is that a user is not forced to learn an algebraic modeling language and can use the usual natural mathematical language, which is learned and practiced by every mathematician, computer scientist, physicist, and engineer.

In addition, this kind of specification of a model is the least error prone, and the most natural way to communicate a model. Once represented in the framework, multiple outputs in different modeling languages (or even descriptions in different natural languages) would not mean extra work for the user if appropriate transformation modules are available.

The MoSMath project makes use of or connects to several already existing software systems:

LaTeX: Being the de facto standard in the mathematical community for decades, the syntax of the input will be a subset of LaTeX.

Markup languages: Texts written in markup languages like XML are highly structured and easily machine readable. We make use of the tool LaTeXML [20] to produce an XML document from a LaTeX input, which can then be translated into records in our data structure.

Algebraic modeling languages: To be able to access a wide variety of solvers, the algebraic modeling language AMPL [7] is used the primary target language.

The Grammatical Framework: A programming language for multilingual grammar applications, which allows us to produce grammatically correct sentences in multiple natural languages [26].

Naproche: An controlled natural language which can be used to interface proof checkers [6].

TPTP: The library "Thousands of problems for theorem provers" provides facilities to interface interactive theorem provers [34].

The thesis of the first author [27] will comprise the main topics of this paper in greater rigor and detail.

2.2 Vision: The FMathL Project

The MoSMath project is embedded into a far more ambitious long-term vision: the FMathL project, described in the extensive FMathL web site.[1] For a summary, see Neumaier and Schodl [24].

[1] The FMathL web site is available at http://www.mat.univie.ac.at/$\sim$neum/FMathL.html

While the MoSMath project creates an interface for optimization problems formulated in almost natural mathematical language, the vision of the FMathL project is an automatic general purpose mathematical research system that combines the indefatigability, accuracy and speed of a computer with the ability to interact at the level of a good mathematics student. Formal models would be specified in FMathL close to the way they would be communicated in a lecture or a paper: with functions, sets, operators, quantifiers, tables and cases rather than loops, indices, diagrams, etc.

FMathL aims at providing mathematical content and proof services as easily as Google provides web services, MATLAB provides numerical services, and MATHEMATICA or MAPLE provide symbolic services. A mathematical assistant based on the FMathL framework should be able to solve standard exercises, intelligently search a universal database of mathematical knowledge, check the represented mathematics for correctness, and aid the author in routine mathematical work. The only extra work the user would have to do is during parse time of the written document, when possible ambiguities have to be resolved.

Important planned features of FMathL include the following:

- It has both a "workbench" character where people store their work locally, as well as a "wiki" character where work can be shared worldwide.
- It *supports* full formalization, but does not *force* it upon the user.
- It incorporates techniques from artificial intelligence.
- It communicates with the user in a natural way.
- The language is extensible, notions can be defined as usual and will then be understood.
- To deal with ambiguities, the system makes use of contextual information.

By complementing existing approaches to mathematical knowledge management, the FMathL project will contribute towards the development of:

- The QED project [3]: FMathL would come with a database of basic mathematics, preferably completely formalized. In addition, the natural interface would make contributing to the QED project easier.
- A universal mathematical database: envisioned, e.g., in Andrews [1] and partially realized in theorem provers with a big library such as MIZAR [35], where they only serve a single purpose.
- An assistant in the sense of Walsh [36] that saves the researcher's time and takes routine work off their shoulders.
- A checker not only for grammar but also for the semantical correctness of mathematical text.
- Automatic translation of mathematical content into various natural languages.

While FMathL reaches far beyond MoSMath, we expect that the framework of the MoSMath project will serve as a first step towards the FMathL project. The FMathL project will also benefit from MoSMath in the sense that once MoSMath is integrated into FMathL, it will make FMathL usable in the restricted domain of optimization long before the full capabilities of FMathL are reached.

2.3 Data Structure: The Semantic Memory

The semantic memory is a framework designed for the representation of arbitrary mathematical content, based on a computer-oriented setting of formal concept analysis [8]. In particular, our goal was to be able to represent mathematical expressions, mathematical natural language, and grammars in a natural way in the semantic memory. We are aware of existing languages and software systems to represent mathematics, but found them inadequate for our goals, see Kofler et al. [14, 15].

We assume an unrestricted set of **objects**. Objects may, but need not have **names**, i.e., alphanumeric strings not beginning with a digit, by which the user refers to an object. `Empty` is the name of an object. Furthermore objects may, but need not have **external values**, i.e., data of arbitrary form, associated with the object, but stored outside the semantic memory.

Variable objects are variables in the usual sense, ranging over the set of objects, but since alphanumeric strings may refer to objects, we refer to variable objects via a string beginning with a hash (`#`) followed by some alphanumeric string. Usually, we will use suggestive strings for variables, e.g., for an object that is intended to be a handle, we use `#handle` or `#h`.

A **semantic mapping** assigns to two objects `#h` and `#f` a unique object `#e` which is `Empty` if `#h` or `#f` is `Empty`. We call an equation of the form `#h.#f=#e` with `#h`, `#f`, and `#e` not `Empty` a **semantic unit** or **sem** . We call `#h` the **handle**, `#f` the **field**, and `#e` the **entry** of the sem. The set of sems with handle `#h` are called the **constituents** of `#h`. Given two sems `#h1.#f1=#e1` and `#h2.#f2=#e2`, if `#e1=#h2` we also write `#h1.#f1.#f2=#e2`, and this notation extends to more dots.

The semantic unit is the smallest piece of information in the semantic mapping. It is intended to be both low level and natural to the human. Depending on the context, the intuitive meaning of a sem `#h.#f=#e` often is "the `#f` of `#h` is `#e`", e.g., `formula27.label=CauchySchwarz` would intuitively mean that the `label` of `formula27` is `CauchySchwarz`. A sem can also express a unary predicate: $P(x)$ could be expressed as `P.x=True`. Since sems are "readable" in this intuitive sense, manipulating information in a semantic mapping is easier and more transparent than in other low level systems like a Turing machine or a register machine.

The semantic mapping codifies the foundations of formal concept analysis in a way suitable for automatic storage and processing of complex records. A statement of the form gIm (interpreted as "the object g has the attribute m") can be represented as a sem `g.m=Present`. The semantic matrix precisely matches *multi valued contexts* [8, p. 36] where I is a ternary relation and $I(g, m, w)$ is interpreted as "the attribute m of object g is w", with the property $I(g, m, w_1)$ and $I(g, m, w_2)$ then $w_1 = w_2$. This corresponds to the sem `g.m=w`, since from `g.m=w1` and `g.m=w2` follows `w1=w2` from the uniqueness of the entry of a semantic mapping.

The semantic memory is also representable within the framework of the semantic web [17]. In particular, we have implemented the semantic memory in RDF [18].

For graphical illustration of a semantic mapping, we interpret a sem `#a.#b=#c` as an edge with label `#b` from node `#a` to node `#c` of a directed labeled graph, called a **semantic graph**. In semantic graphs, named objects are represented by their names, and objects that have no name will be represented by automatically generated strings preceded by a dollar sign (`$`). In a semantic graph, objects that have an external value are printed as a box containing that value. For better readability we use dashed edges for edges labeled with `type`, since these sems have importance for typing.

A **record** with handle `#rec` is the set of sems `#h.#f=e` reachable from `#rec` in the sense that there exist objects `#f1`,...,`#f`n such that `#rec.#f1`$\cdots$`.#f`n`=#h`. In contrast to records in programming languages such as Pascal, records in a semantic mapping may have cycles since back references are allowed; this is necessary to encode grammars in a natural way.

For example, the expression

$$\lambda x.x + 1$$

may be represented by the record with handle `$2472` shown in Fig. 2.1. Further examples can be found in Appendix 2.9, see also Schodl and Neumaier [31].

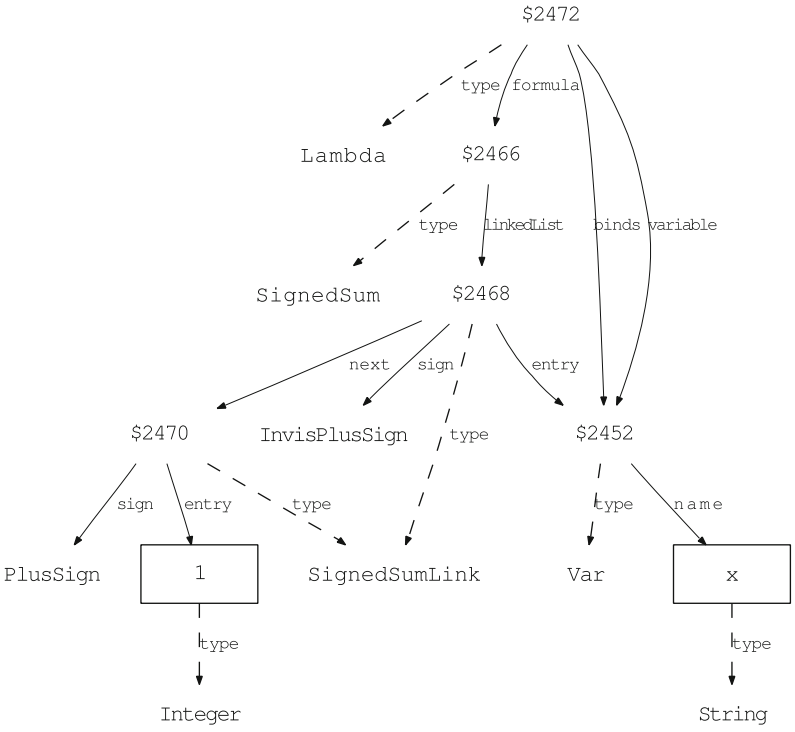


Fig. 2.1 A record representing an the expression $\lambda x.x + 1$

Semantic mappings are used to store mathematics. In order to work on data represented, the framework needs to be dynamic. The data structure of the semantic mapping that changes over time is called the **semantic memory** (SM).

While not identical, our representation shares features with some existing representation frameworks:

- The need to represent (mathematical) natural language poses the requirement of "structure sharing", i.e., a phrase, an expression etc. only has to be represented once while it may occur multiple times in the text. This suggests a graph structure rather than a tree structure, as facilitated many knowledge representation system, e.g., in SNePS [33]. SNePS also makes use of a labeled graph, but on the other hand uses "structured variables", storing quantification and constraints together with the variable. This is not desirable when representing mathematics since structured variables make it complicated to represent the difference between, e.g.,

$$\forall x \forall y (P(x, y) \Longrightarrow G(x)) \qquad \text{and} \qquad \forall x ((\forall y P(x, y)) \Longrightarrow G(x)).$$

- The record structure where a complex record is built up from combining more elemental records is similar to a parse tree, especially to a parse tree for a dependency grammar [5]. However, a parse tree is always a tree and does not allow structure sharing.

Context dependence.

Even on the lowest level of information representation, context dependence is already incorporated in a natural way.

The same object $\#\circ$ may be reachable via different paths, and while it is the same object, the access is different and so can be its interpretation. Therefore the path how an object is accessed contains information about the context, and about the usage of the object. The additional information contained in the path can be used to account for some of the contextual information in natural languages: while natural languages are to a large extent context free [11], the semantics is also based on the context.

A logic of context compatible with the present framework is presented in Neumaier and Marginean [23].

2.4 Processing: The Semantic Virtual Machine

We define a virtual machine that operates on the SM called the **semantic virtual machine** (SVM) to be able to rigorously argue about processes in the SM, and to be able to proof properties. This abstract machine can be implemented in many ways; we currently have implementations in Matlab (using a sparse matrix to represent the SM) and in C++/Soprano (using RDF).

The semantic memory of the SVM contains a **program** to execute, its **context** (i.e., input and output, corresponding to the tape of an ordinary Turing machine), and the information about **flow control** as well. To enable the processing of more than one program in the same memory each program has its own **core**, i.e., a record reserved for temporary data. Since the core is the most important record for a program we use the **caret** `^` to abbreviate the reference the core of the program. Hence `^a` means `#core.a`, where `#core` is the core of the program under consideration.

The most elementary part of the SVM programming language is a command. A **process** is a sequence of commands, beginning with the command `process #proc`. Every process ends with a command that either halts the SVM or calls another process. An **SVM program** is the command `program #prog` followed by a sequence of processes. The (changing) SVM command currently executed is called the **focus**. Each SVM program is completely represented in the SM. When invoked, the SVM is given a record containing a program, an object it can use as its core.

The SVM has the ability to access the facilities of the physical device it is implemented on. This may provide the SVM with much better performance for tasks it can export, and allows the use of trusted external processors and programs in different programming languages.

External values can be copied to the memory of the SVM, and conversely. This is done by the commands `in` and `out`. The information about how to represent the external value in the memory of the SVM is called the **protocol**, and is used as an argument for the commands `in` and `out`. Our current implementation includes protocols for representing natural numbers, character strings, SVM programs and tapes and transition tables for Turing machines. There may be an arbitrary number of protocols, as long as the device the SVM is implemented on knows how to interpret them.

The commands of the SVM language fall into four groups: Table 2.1 describes the commands that are needed to give the program an appropriate structure. Table 2.2 contains the assignments, i.e., those commands that perform alterations in the SM. Table 2.3 gives the commands used for flow control, and Table 2.4 the commands that establish communication with the physical device, namely call external processes and access external values. If an assignment refers to a non existing node, an error is produced. For convenience, some commands also appear in variants that could be simulated by a sequence of other commands.

In the informal description of commands given here, `VALUE(#obj)` refers to the external value of the object `#obj`. The effect of each command is formally defined by an operational semantics given in Schodl [27].

Details of the SVM are discussed in Neumaier and Schodl [25].

Table 2.1 Structuring commands

SVM command	Comment
`program #1`	first line of the program `#1`
`process #1`	first line of the process `#1`
`start #1`	start with process `#1`

Table 2.2 Assignment commands

SVM command	Comment
`^#1=(^#2==^#3)`	sets `^#1` to `True` if `^#2` = `^#3`, else to `False`
`^#1.#2=^#3`	assigns `^#3` to `^#1.#2`
`^#1.^#2=^#3`	assigns `^#3` to `^#1.^#2`
`^#1.#2=const #3`	assigns `#3` to `^#1.#2`
`^#1=^#2.#3`	assigns `^#2.#3` to `^#1`
`^#1=^#2.^#3`	assigns `^#2.^#3` to `^#1`
`^#1=fields of ^#2`	assigns the used fields of the record `^#2` to `^#1.1`, `^#1.2`,...
`^#1=exist(#2.#3)`	sets `^#1` to `True` if `#1.#2` exists, else to `False`
`^#1=exist(^#2.^#3)`	sets `^#1` to `True` if `^#1.^#2` exists, else to `False`

Table 2.3 Commands for flow control

SVM command	Comment
`goto #1`	sets the focus to the first line of process `#1`
`goto ^#1`	sets the focus to the first line of process `^#1`
`if ^#1 goto #2`	sets the focus to the first line of process `#2` if `^#1=True`, and to the next line if `^#1=False`
`stop`	ends a program

Table 2.4 Commands for external communication

SVM command	Comment
`external #1(^#2)`	starts execution of the external processor `#1` with context `^#2`
`external ^#1(^#2)`	starts execution of the external processor `^#1` with context `^#2`
`in ^#1 as #2`	imports `VALUE(^#1)` into a record with handle `^#1` using protocol `#2`
`in ^#1 as ^#2`	imports `VALUE(^#1)` into a record with handle `^#1` using protocol `^#2`
`out ^#1 as #2`	exports the record with handle `^#1` into `VALUE(^#1)` using protocol `#2`
`out ^#1 as ^#2`	exports the record with handle `^#1` into `VALUE(^#1)` using protocol `^#2`

As a test case and to show Turing completeness, we programmed a Turing machine in SVM. The SVM program that simulates a Turing machine contains 71 commands and can be found in Neumaier and Schodl [25]. Since an ordinary Turing machine has no external storage and it is not specified how an external processor should behave, it is impossible to give an ordinary Turing machine that simulates an arbitrary SVM program.

2.5 Representation: The Type System

An essential step that brings formal structure into the semantic memory is the introduction of **types**. In order to represent mathematics specified in a controlled natural language, a concept of typing is needed that is more general than traditional type systems. It must cover and check for well-formedness of both structured

records as commonly used in programming languages and structures built from linguistic grammars. In particular each grammatical category must be representable as a type, in order to provide a good basis for the semantical analysis of mathematical language.

Information in the SM is organized in records. When using a record, or passing it to some algorithm, we need information about the structure of this record. Since we do not want to examine the whole graph every time, we assign types, both to objects and to sems.

For a detailed discussion of the concepts and rigorous definitions, see Schodl and Neumaier [32], illustrations and examples are given in Schodl and Neumaier [31].

Types can be defined using plain text documents called **type sheets**. An example of a type sheet can be found in Schodl and Neumaier [29]. Tables 2.5 and 2.6 gives an overview of the operators in a type sheet and their usage. For many tasks, giving an (annotated) type sheet defines the syntax of an arbitrary construction in the SM, and in many cases it even suffices to define the semantics.

Our form of type inheritance adds the specifications from an existing declared type `#T` to a newly defined declared type `#t`. In this case, we call `#T` the **template** of `#t`. All the requirements from `#T` apply to `#t`, and additionally the requirements for `#t`. The difference to the inheritance in typical programming languages is that:

- The declared type and the template may pose requirements on the same constituent.
- Inheritance from multiple declared types is possible, but there is always only at most one template of a declared type, and
- It is stored that the template of `#D` is `#T`.

Table 2.5 Keywords in declarations of unions and atomics

Operator	Arguments	Usage
`nothing`	None	Defines an atomic type
`union`	List of names	Defines a union
`atomic`	List of names	Defines a union of atomic types
`complete`	None	Closes a union

Table 2.6 Keywords in declarations of proper types

Operator	Arguments	Usage
`allOf`	List of equations	Restricts entry of certain fields
`oneOf`	List of equations	Restricts entry of certain fields
`someOf`	List of equations	Restricts entry of certain fields
`optional`	List of equations	Restricts entry of certain fields
`fixed`	List of equations	Restricts entry of certain fields
`only`	List of equations	Restricts entry of certain fields
`someOfType`	List of equations	Restricts entry of certain fields
`itself`	List of names	Restricts entry of certain fields
`array`	List of equations	Restricts entry of certain fields
`index`	List of equations	Requires to index each instance
`template`	One name	Assigns a template
`nothingElse`	None	Forbids further fields

The type is always assigned with respect to a specified type system. A **type system** specifies the **categories** belonging to it, and their properties. The object `Empty` is never a category. In every type system, the set of categories is ordered by an irreflexive and transitive partial order relation <. If `#C1<#C2`, we say that `#C2` **contains** `#C1` in this type system. We write $\leq$ for the associated reflexive partial order, with `#C1`$\leq$`#C2` iff `#C1<#C2` or `#C1=#C2`.

A category is called a **type** if it is minimal in the ordering <, and a **union** otherwise. Types come in three forms: the **default type** `Object`, **atomic types**, and **proper types**. All categories except `Object` are declared in the record whose handle is the type system. This record can be created by means of a type sheet, a piece of text containing the specifications.

If `#a.#f=Empty` for every field `#f`, then `#a` is an **atomic object**. Atomic objects are used as objects with a fixed semantic meaning. Objects of a proper type always have a field `type` whose entry is this proper type. Proper types are used to pose requirements on the other constituents of the objects of this type.

We define the **type of object** `#obj`:

If `#obj.type` is a declared type, then the type of `#obj` is `#obj.type`.

If `#obj.type=Empty` and `#obj` is an atomic type, then the type .

of `#obj` is `#obj`

Otherwise, the type of `#obj` is `Object`.

Hence every object has a type. Determining the type of an object has complexity $O(1)$.

An object `#obj` **matches** a category `#C` in the type system `#TS` if either `#t<#C` (in the type system `#TS`) where `#t` is the type of `#obj`, or `#C=Object`.

The definition of well-typedness of a record is more intricate and will only be sketched here.

A pair (`#o`,`#f`) of two objects is called a **position**. A type sheet poses requirements on certain positions, called **declared positions**. Roughly speaking, if for all declared positions (`#o`,`#f`) in a record, the object `#o.#f` matches the required type, then the record is **well-typed**, and **ill-typed** otherwise. Whether or not the record is well-typed can be checked in time linear in the size of the record.

We represent type declarations and unions as records, in order that a type checker, working on the semantic memory, has all the information it needs in the semantic memory and does not need any external type sheet. Thus we store a type declaration as a well-typed record in the SM. A type declaration `#TD` has `#TD.type=Type`.

Typing in FMathL and typing in XML bear significant similarities, most notably with DTD and Relax NG. For a description of DTD, Relax NG and other XML schemas, see Lee and Chu [16]. Some of the operators in type declarations have a direct correspondence in the language of DTD and the RelaxNG compact syntax. E.g., `?` in DTD corresponds to `optional`, the pipe | corresponds to `oneOf` and parentheses `( )` correspond to `allOf`. A valid XML document corresponds to a well-typed record. However, there are also important differences, since cycles are an important feature of our framework that enables an efficient representation of concrete and abstract grammars, while XML documents are always organized in trees.

2.6 Higher Level Processing: CONCISE

The SVM is merely intended for definition, checkability and low level implementation. After some experience with SVM programs we designed a programming environment that is intended for user-friendly data entry and manipulation, algorithm design and execution, and more general for interaction with the semantic memory. Loosely speaking, it is an integrated development environment (IDE) for mathematics in the semantic memory. This environment is called CONCISE.

A Java implementation of CONCISE is being written by Ferenc Domes, and publication will be announced at the FMathL web site.[2] It consists of a versatile GUI (graphical user interface) that enables the user to view, create and manipulate sems and records in a natural way. An interpreter for a Turing complete subset of CONCISE written for the SVM only requires 330 lines of SVM code.

Algorithms can be programmed and executed in CONCISE, but algorithms are represented as records in the SM, making CONCISE a text-free programming environment. Nevertheless, for debugging and alternative coding there are text views on CONCISE programs.

CONCISE has configurable display and text completion, and will support types, function calls, different kinds of variables (global, local, static and persistent), loops over all fields of an object, multiple users and multiple languages.

CONCISE will also incorporate a parser capable of dealing with dynamically changing grammars. This is necessary because in many specifications in ordinary mathematical language, the syntax (and hence the grammar) is partially defined through the context. In particular, definitions give not only the semantics of the term being defined, but implicitly also its grammatical function.

2.7 Mathematical Modeling

The semantic memory is designed for representing and processing mathematical content. While generality of the representation was one important goal, another one was to be able to run algorithms on the records in a transparent way.

To test the practicability of the present framework, mathematical content from different sources is represented in the semantic memory:

- A significant fraction of the optimization problems from the OR Library [2] were represented manually in the semantic memory. This is the most important application for the MOSMATH project. We designed a natural representation of these optimization problems as records, in order to be able to run algorithms on these records. The representation is defined in a type sheet [29]. There are algorithms that produce LaTeX from formulas and whole optimization problems.

[2] http://www.mat.univie.ac.at/$\sim$neum/FMathL.html

Another algorithm enriches the representation of optimization problems in the semantic memory such that an AMPL document specifying a valid, numerically solvable model can be produced.

- Different types of mathematical formulas were extracted from lecture notes about basic analysis and linear algebra [21]. These were manually fed into the semantic memory to assure generality of the representation of formulas [31]. Partial work on the grammar of the text part of the lecture notes can be found in Schodl and Neumaier [28]. Some of the expressions from the lecture notes are presented in Sect. 2.9.
- An interface was written to automatically import formulas from the TPTP library [34], "Thousands of Problems for Theorem Provers", a library of formulas for theorem provers, taken from different branches of mathematics.
- An interface was written to automatically import formalized proofs written in the controlled natural language of Naproche [6].

Grammatical issues in the translation from mathematical language into SM documents, and from SM records to natural language, including a dynamic parser for parallel multiple context free grammars (PMCFGs) and an interface to the "Grammatical Framework" [26] are the subject of the PhD thesis by Kofler (in preparation, [13]). This parser will handle updates to the grammar, a feature necessary to handle mathematical definitions that introduce new syntax.

Example: Knapsack problem

The multidimensional knapsack problem is a standard optimization problem, and also contained in the OR-Library [2]. A possible natural formulation is:

Let the integer N be the number of contracts, let the integer M be the number of budgets. Let c_j be the contract volume of project j for $j = 1, \ldots, N$, let $A_{i,j}$ be the estimated cost of budget i for project j for $i = 1, \ldots, M$ and $j = 1, \ldots, N$, and let B_i be the available amount of budget i for $i = 1, \ldots, M$. For $j = 1, \ldots, N$, let $x_j = 1$ if project j is selected, and let $x_j = 0$ otherwise.
Problem : Given integers N and M, vector c, matrix A and vector B, find the binary vector x such that

$$\sum_{j=1}^{N} c_j \, x_j$$

is maximal under the constraint $\sum_{j=1}^{N} A_{i,j} \, x_j \leq B_i$ for $i = 1, \ldots, M$.

When represented in the semantic memory, a transformation module produces a LaTeX-document that is identical to the model description above, except for

grammatical errors. Another transformation module produces from the same record in the semantic matrix the following AMPL model.

```
param N ;
param M ;
param c{j in 1..N} ;
param A{i in 1..M , j in 1..N} ;
param B{i in 1..M} ;
var x{j in 1..N} binary ;

maximize target : sum{j in 1..N}(c[j] *
x[j]);
subject to constraint_3014{i in 1..M}
: sum{j in 1..N}(A[i , j] * x[j]) <=
B[i];
```

2.8 Reflection

In the most general sense, reflection is the representation of parts of a system within itself.

To test the power of our setting, we tried to achieve a multiple reflection of various features on levels as low as possible:

- For the SVM, reflection is giving an SVM program that can simulate every other SVM program in the same way a universal Turing machine simulates an ordinary Turing machine. We call this program the **universal semantic virtual machine** (USVM). The context of the USVM contains the SVM program `P` and the context of `P`. When the USVM has finished, the USVM has produced the same changes in the context as `P` would have produced when called directly. The USVM is a program short and transparent enough to be checked by hand. It has only 164 commands; compare this, e.g., to the reflective interpreter by Jefferson and Friedman [10], which has 273 lines and the implementation by McCarthy [19] of a Lisp function evaluation in Lisp which has 58 (but very complex) commands. The USVM serves two purposes: it is a reflection of the actions of the SVM commands, and it gives us a check on many aspects of the SVM for correctness. Once one has convinced oneself of the correctness of the USVM, one can make the implementation of the SVM on some physical device also trustworthy by checking empirically (or, in principle, in a formal way) that any SVM program executed by the implemented SVM produces the same output as in the case when the USVM simulates this program. The type sheet for types [30] has only 40 lines, compared to the RelaxNG schema [4], its analogon in RelaxNG, which has over 300 lines, or Meta-DSD [12], its analogon in DSD, which has over 700 lines.
- The type system of CONCISE can be formulated as a type sheet, and the typechecker itself can be implemented as a CONCISE program.

- An interpreter of the programming language of CONCISE as an SVM program. This makes a large part of CONCISE independent of any specific implementation, since merely the SVM needs to be implemented.
- A definition of the semantics of mathematical concepts, represented within the SM.
- An operational semantics for the SVM, i.e., a rigorous description of the action of the SVM in mathematical terms, to be represented later in a reflective fashion on the concept level of CONCISE.
- For the envisioned FMathL, reflection is a process to deal with the distinction between object level and metalevel which occurs in every foundation of mathematics, hence also in FMathL. Here reflection means the process to express the notions of the metalevel as objects on the object level, and then formulate the relations between these notions as relations between objects. The same process of reflection of the metalevel L_0 in the object level L_1 can be repeated as a reflection of level L_k into the level L_{k+1}. This way, the metalevel is never fully formalized, but if the reflection step L_k in L_{k+1} can be accomplished without running into paradoxes or insufficiencies of the language, we can be confident of the adequacy of our foundation. For more details, and for the significance of reflection for FMathL as a foundation of mathematics, see Neumaier [22].

2.9 Examples of Expressions

We give a few examples for the representation of expressions in the SM as records, see Schodl and Neumaier [31] for many more.

The following table gives some statistics of the examples.

Example	# visible symbols	# LaTeX-characters	# sems
0	14	49	34
1	22	45	53
2	11	92	48

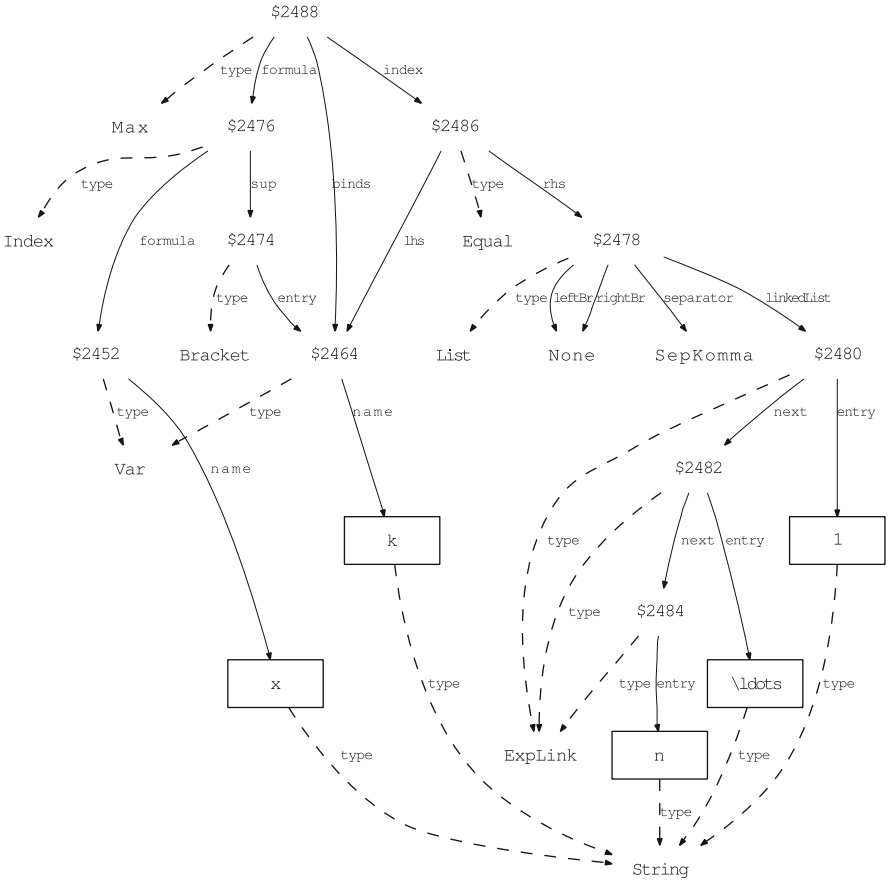
For each example, we give the rendered expression followed by a table containing the sems and external values in the left column, and the MATLAB construction code used to write the sems into our MATLAB implementation of the semantic memory in the right column. This is followed by a semantic graph of the record.

Example 0.

$$\max_{k=1,\dots,n} x^{(k)}$$

```
\max_{k{=} 1 , \ldots , n }{{}{x}^{\left(k\right)}}
```

record	MATLAB construction code
$2488.type=max	x=createvar('x');
$2488.formula=$2476	k=createvar('k');
$2488.binds=$2464	listk=create('VarList',{k});
$2488.index=$2486	n=createstring('n');
$2464.type=Var	one=createstring('1');
$2464.name=$2466	dots=createstring('\ldots');
$2466.type=String	br=create('Bracket',k);
$2476.type=Index	in=create('Script',x,[],br);
$2476.formula=$2452	komma=createname('SepKomma');
$2476.sup=$2474	none=createname('None');
$2452.type=Var	lst=create('List',{one,dots,n},none,komma,none);
$2452.name=$2454	eq=create('Equal',k,lst);
$2454.type=String	ex=create('Max',in,listk,eq);
$2474.type=Bracket	
$2474.entry=$2464	
$2486.type=Equal	
$2486.lhs=$2464	
$2486.rhs=$2478	
$2478.type=List	
$2478.leftBr=None	
$2478.separator=SepKomma	
$2478.rightBr=None	
$2478.linkedList=$2480	
$2480.type=ExpLink	
$2480.next=$2482	
$2480.entry=$2470	
$2470.type=String	
$2482.type=ExpLink	
$2482.next=$2484	
$2482.entry=$2472	
$2472.type=String	
$2484.type=ExpLink	
$2484.entry=$2468	
$2468.type=String	
VALUE($2454) = x	
VALUE($2466) = k	
VALUE($2468) = n	
VALUE($2470) = 1	
VALUE($2472) = \ldots	

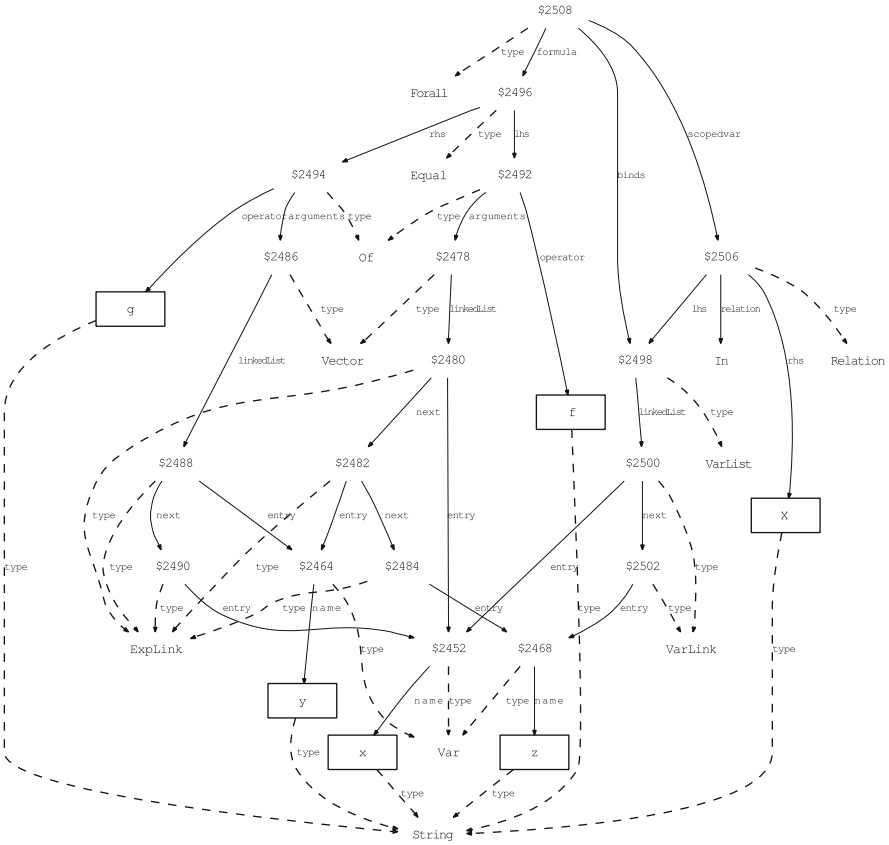


Example 1.

$$\forall x, z \in X : f\,(x, y, z) = g\,(y, x)$$

```
\forall x , z \in X : f \left(x , y , z\right) {=}
g \left(y , x\right)
```

record		MATLAB construction code
$2508.type=Forall	$2470.type=String	x=createvar('x');
$2508.formula=$2496	$2494.type=Of	y=createvar('y');
$2508.binds=$2498	$2494.operator=$2474	z=createvar('z');
$2508.scopedvar=$2506	$2494.arguments=$2486	f=createstring('f');
$2496.type=Equal	$2474.type=String	g=createstring('g');
$2496.lhs=$2492	$2486.type=Vector	X=createstring('X');
$2496.rhs=$2494	$2486.linkedList=$2488	xyz=create('Vector',{x,y,z});
$2492.type=Of	$2488.type=ExpLink	yx=create('Vector',{y,x});
$2492.operator=$2472	$2488.next=$2490	lhs=create('Of',f,xyz);
$2492.arguments=$2478	$2488.entry=$2464	rhs=create('Of',g,yx);
$2472.type=String	$2490.type=ExpLink	eq=create('Equal',lhs,rhs);
$2478.type=Vector	$2490.entry=$2452	xz = create('VarList',{x,z});
$2478.linkedList=$2480	$2498.type=VarList	in=createname('In');
$2480.type=ExpLink	$2498.linkedList=$2500	dummy=create('Dummy',x);
$2480.next=$2482	$2500.type=VarLink	var=create('Relation',xz,in,X);
$2480.entry=$2452	$2500.next=$2502	ex=create('Forall',eq,var,xz);
$2452.type=Var	$2500.entry=$2452	
$2452.name=$2454	$2502.type=VarLink	
$2454.type=String	$2502.entry=$2468	
$2482.type=ExpLink	$2506.type=Relation	
$2482.next=$2484	$2506.lhs=$2498	
$2482.entry=$2464	$2506.rhs=$2476	
$2464.type=Var	$2506.relation=In	
$2464.name=$2466	$2476.type=String	
$2466.type=String	VALUE($2454) = x	
$2484.type=ExpLink	VALUE($2466) = y	
$2484.entry=$2468	VALUE($2470) = z	
$2468.type=Var	VALUE($2472) = f	
$2468.name=$2470	VALUE($2474) = g	
	VALUE($2476) = X	

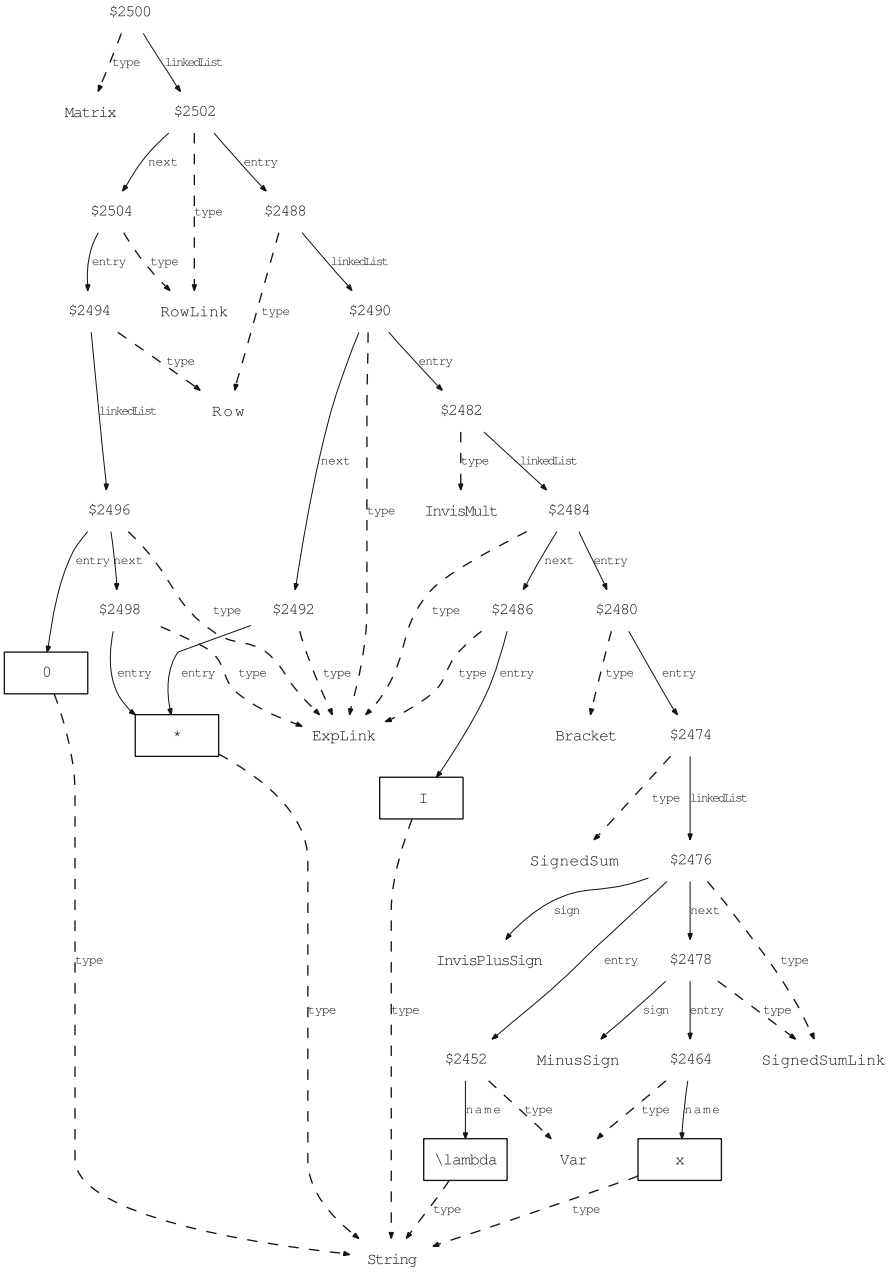


Example 2.

$$\left(\begin{array}{cc} \left(\lambda - x \right) I & * \\ 0 & * \end{array} \right)$$

```
\left(\begin{array}{cc} \left( \lambda - x \right) I & * \\
 0 & * \\ \end{array} \right)
```

record	MATLAB construction code
$2500.type=Matrix	lambda=createvar('\lambda');
$2500.linkedList=$2502	x=createvar('x');
$2502.type=RowLink	id=createstring('I');
$2502.next=$2504	ast=createstring('*');
$2502.entry=$2488	zero=createstring('0');
$2488.type=Row	iplus=createname('InvisPlusSign');
$2488.linkedList=$2490	minus=createname('MinusSign');
$2490.type=ExpLink	mi=create('SignedSum',{{iplus,lambda},{minus,x}});
$2490.next=$2492	bra=create('Bracket',mi);
$2490.entry=$2482	mult=create('InvisMult',{bra,id});
$2482.type=InvisMult	row1=create('Row',{mult,ast});
$2482.linkedList=$2484	row2=create('Row',{zero,ast});
$2484.type=ExpLink	ex=create('Matrix',{row1,row2});
$2484.next=$2486	
$2484.entry=$2480	
$2480.type=Bracket	
$2480.entry=$2474	
$2474.type=SignedSum	
$2474.linkedList=$2476	
$2476.type=SignedSumLink	
$2476.next=$2478	
$2476.entry=$2452	
$2476.sign=InvisPlusSign	
$2452.type=Var	
$2452.name=$2454	
$2454.type=String	
$2478.type=SignedSumLink	
$2478.entry=$2464	
$2478.sign=MinusSign	
$2464.type=Var	
$2464.name=$2466	
$2466.type=String	
$2486.type=ExpLink	
$2486.entry=$2468	
$2468.type=String	
$2492.type=ExpLink	
$2492.entry=$2470	
$2470.type=String	
$2504.type=RowLink	
$2504.entry=$2494	
$2494.type=Row	
$2494.linkedList=$2496	
$2496.type=ExpLink	
$2496.next=$2498	
$2496.entry=$2472	
$2472.type=String	
$2498.type=ExpLink	
$2498.entry=$2470	
VALUE($2454) = \lambda	
VALUE($2466) = x	
VALUE($2468) = I	
VALUE($2470) = *	
VALUE($2472) = 0	



Acknowledgements Support by the Austrian Science Fund (FWF) under contract number P20631 is gratefully acknowledged.

References

1. Andrews, P.: A Universal Automated Information System for Science and Technology. In: First Workshop on Challenges and Novel Applications for Automated Reasoning, pp. 13–18 (2003)
2. Beasley, J.: OR-Library: Distributing test problems by electronic mail. Journal of the Operational Research Society **41**(11), 1069–1072 (1990)
3. Boyer, R., et al.: The QED Manifesto. Automated Deduction–CADE **12**, 238–251 (1994)
4. Clark, J., Murata, M., et al.: Relax NG specification – Committee Specification 3 December 2001. Web document (2001). http://www.oasis-open.org/committees/relaxng/spec-20011203.html
5. Covington, M.: A fundamental algorithm for dependency parsing. In: Proceedings of the 39th annual ACM southeast conference, pp. 95–102. Citeseer (2001)
6. Cramer, M., Fisseni, B., Koepke, P., Kühlwein, D., Schröder, B., Veldman, J.: The Naproche Project Controlled Natural Language Proof Checking of Mathematical Texts. Controlled Natural Language pp. 170–186 (2010)
7. Fourer, R., Gay, D., Kernighan, B.: A modeling language for mathematical programming. Management Science **36**(5), 519–554 (1990)
8. Ganter, B., Wille, R.: Formale Begriffsanalyse: Mathematische Grundlagen. Springer-Verlag Berlin Heidelberg New York (1996)
9. Humayoun, M., Raffalli, C.: MathNat – Mathematical Text in a Controlled Natural Language. Special issue: Natural Language Processing and its Applications p. 293 (2010)
10. Jefferson, S., Friedman, D.: A simple reflective interpreter. LISP and symbolic computation **9**(2), 181–202 (1996)
11. Jurafsky, D., Martin, J., Kehler, A., van der Linden, K., Ward, N.: Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition, vol. 163. MIT Press (2000)
12. Klarlund, N., Moeller, A., I., S.M.: Meta-DSD. Web document (1999). http://www.brics.dk/DSD/metadsd.html
13. Kofler, K.: A Dynamic Generalized Parser for Common Mathematical Language. PhD thesis (In preparation)
14. Kofler, K., Schodl, P., Neumaier, A.: Limitations in Content MathML. Technical report (2009). http://www.mat.univie.ac.at/~neum/FMathL.html#Related
15. Kofler, K., Schodl, P., Neumaier, A.: Limitations in OpenMath. Technical report (2009). http://www.mat.univie.ac.at/~neum/FMathL.html#Related
16. Lee, D., Chu, W.: Comparative analysis of six XML schema languages. ACM SIGMOD Record **29**(3), 76–87 (2000)
17. Lee, T., Hendler, J., Lassila, O., et al.: The semantic web. Scientific American **284**(5), 34–43 (2001)
18. Manola, F., Miller, E., et al.: RDF Primer. Web document (2004). http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
19. McCarthy, J.: A micro-manual for LISP – not the whole truth. ACM SIGPLAN Notices **13**(8), 215–216 (1978)
20. Miller, B.: LaTeXML the manual. Web document (2011). http://dlmf.nist.gov/LaTeXML/manual.pdf
21. Neumaier, A.: Analysis und lineare Algebra. Lecture notes (2008). http://www.mat.univie.ac.at/~neum/FMathL.html#ALA
22. Neumaier, A.: The FMathL mathematical framework. Draft version (2009). http://www.mat.univie.ac.at/~neum/FMathL.html#foundations
23. Neumaier, A., Marginean, F.A.: Models for context logic. Draft version (2010). http://www.mat.univie.ac.at/~neum/FMathL.html#Contextlogic
24. Neumaier, A., Schodl, P.: A Framework for Representing and Processing Arbitrary Mathematics. Proceedings of the International Conference on Knowledge Engineering and Ontology

Development pp. 476–479 (2010). An ealier version is available at http://www.mat.univie.ac.at/~schodl/pdfs/IC3K_10.pdf
25. Neumaier, A., Schodl, P.: A semantic virtual machine. Draft version (2011). http://www.mat.univie.ac.at/~neum/FMathL.html#SVM
26. Ranta, A.: Grammatical framework. Journal of Functional Programming **14**(02), 145–189 (2004)
27. Schodl, P.: Foundations for a self-reflective, context-aware semantic representation of mathematical specifications. PhD thesis (2011)
28. Schodl, P., Neumaier, A.: An experimental grammar for German mathematical text. Manuscript (2009). http://www.mat.univie.ac.at/~neum/FMathL.html#ALA
29. Schodl, P., Neumaier, A.: A typesheet for optimization problems in the semantic memory. Web document (2011). Available at http://www.mat.univie.ac.at/~neum/FMathL.html#TypeSystems
30. Schodl, P., Neumaier, A.: A typesheet for types in the semantic memory. Web document (2011). Available at http://www.mat.univie.ac.at/~neum/FMathL.html#TypeSystem
31. Schodl, P., Neumaier, A.: Representing expressions in the semantic memory. Draft version (2011). http://www.mat.univie.ac.at/~neum/FMathL.html#TypeSystem
32. Schodl, P., Neumaier, A.: The FMathL type system. Draft version (2011). http://www.mat.univie.ac.at/~neum/FMathL.html#TypeSystem
33. Shapiro, S.: An introduction to SNePS 3. Conceptual Structures: Logical, Linguistic, and Computational Issues pp. 510–524 (2000)
34. Sutcliffe, G., Suttner, C.: The TPTP problem library. Journal of Automated Reasoning **21**(2), 177–203 (1998)
35. Trybulec, A., Blair, H.: Computer assisted reasoning with Mizar. In: Proceedings of the 9th International Joint Conference on Artificial Intelligence, pp. 26–28. Citeseer (1985)
36. Walsh, T.: A Grand Challenge for Computing Research: a mathematical assistant. In: First Workshop on Challenges and Novel Applications for Automated Reasoning, pp. 33–34 (2003)

Part II
Selected Algebraic Modeling Systems

This second and main part of the book illuminates selected algebraic modeling systems. It covers CPLEX Studio (formerly, OPL Studio), GAMS, Mosel, and ZIMPL in great detail. Those chapters reflect the personal views and focus of the authors. It is left to the reader to compare the languages and see which one serves his personal needs best. The GAMS contribution focuses on the purposes of modeling languages with respect to different groups of models or customers, respectively, and describes a few special features of this modeling language. More on language elements related to constraint programming is found in the CPLEX Studio and Mosel chapters. A useful property in the modeling language Moselis its concept of *modularity*, which makes it possible to extend this language according to one's needs and to provide new functionality, in particular to deal with other types of problems and solvers.

Chapter 3
GUSS: Solving Collections of Data Related Models Within GAMS

Michael R. Bussieck, Michael C. Ferris, and Timo Lohmann

Abstract In many applications, optimization of a collection of problems is required where each problem is structurally the same, but in which some or all of the data defining the instance is updated. Such models are easily specified within modern modeling systems, but have often been slow to solve due to the time needed to regenerate the instance, and the inability to use advance solution information (such as basis factorizations) from previous solves as the collection is processed. We describe a new language extension, GUSS, that ***g***athers data from different sources/symbols to define the collection of models (called scenarios), ***u***pdates a base model instance with this scenario data and ***s***olves the updated model instance and ***s***catters the scenario results to symbols in the GAMS database. We demonstrate the utility of this approach in three applications, namely data envelopment analysis, cross validation and stochastic dual dynamic programming. The language extensions are available for general use in all versions of GAMS starting with release 23.7.

3.1 Introduction

Algebraic modeling systems (such as GAMS, AMPL, AIMMS, etc) are a well established methodology for solving broad classes of optimization problems arising in a wide variety of application domains. These modeling systems take data from

M.R. Bussieck (✉)
GAMS Software GmbH, Cologne
e-mail: mbussieck@gams.com

M.C. Ferris
University of Wisconsin, Madison, Wisconsin, USA
e-mail: ferris@cs.wisc.edu

T. Lohmann
Colorado School of Mines, Golden, Colorado
e-mail: tlohmann@mines.edu

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_3, © Springer-Verlag Berlin Heidelberg 2012

a rich collection of sources including databases, spreadsheets, web interfaces and simple text files and are able to process specific instantiations of the often large and complex models using state of the art solution engines.

We do not decribe the basics of the GAMS language in this paper but refer to the GAMS paper in the first volume in this series [18] published 8 years ago. In that paper [7] the authors focused on the basic ideas behind GAMS and algebraic modeling in general and highlighted features unique to GAMS. Since 2003, the GAMS system has continuously improved. Maintaining a commercial algebraic modeling system requires to stay current with new developments in optimization, computer science and software engineering. Release notes, available at http://www.gams.com/docs/release/release.htm, provide a minute list of new GAMS features. Besides improved on-line documentation (McCarl Guide), utilities and component libraries for data exchange and control of GAMS runs, the most visible improvements to the GAMS language are a new macro facility, enhancements to the compile time syntax, and the support of external function libraries. As a solver independent modeling system, GAMS provides seamless integration of new solver developments. Deterministic global optimization solvers have been added to the system as well as the free COIN-OR solver suite and the latest addition of a commercial strength LP/MIP solver: GUROBI. Support for in-core and parallel execution of solves utilizing grid computing technology has been added. New model paradigms combining and extending traditional optimization and complementarity models have emerged over the last years. The recently introduced GAMS/EMP framework (Extended Mathematical Programming) helps to foster research on models, reformulations, and algorithms and provides a platform to disseminate results into real world applications.

The purpose of this paper is to detail a recent extension of the GAMS modeling system that allows collections of models (parameterized exogenously by a set of samples or indices) to be described, instantiated and solved efficiently.

As a specific example, we consider the parametric optimization problem $\mathscr{P}(s)$ defined by:

$$\min_{x \in X(s)} f(x;s) \text{ s.t. } g(x;s) \leq 0 \tag{3.1}$$

where $s \in S = \{1, \ldots, K\}$. Note that each scenario s represents a different problem for which the optimization variable is x. The form of the constraint set as given above is simply for concreteness; equality constraints, range and bound constraints are trivial extensions of the above framework. Clearly the problems $\mathscr{P}(s)$ are interlinked and we intend to show how such problems can be easily specified within GAMS and detail one type of algorithmic extension that can exploit the nature of the linkage. Other extensions of GAMS allow solves to be executed in parallel or using grid computing resources [6].

The paper is organized as follows. In Sect. 3.2 we outline the motivation and design of the system for describing and solving $\mathscr{P}(s)$ and give some overview of the available options. Section 3.3 gives three examples of the use of this methodology to the problems of data envelopment analysis (DEA—Sect. 3.3.1), cross validation for problems in machine learning (Sect. 3.3.2) and finally to stochastic dual dynamic

programming (SDDP—Sect. 3.3.3). Note that in our description we will use the terms indexed, parameterized and scenario somewhat interchangeably.

Furthermore we assume that the reader has basic knowledge about the modeling language GAMS.

3.2 Design Methodology

One of the most important functions of GAMS is to build a model instance from the collection of equations (i.e. an optimization model defined by the GAMS keyword MODEL) and corresponding data (consisting of the content of GAMS (sub)sets and parameters). Such a model instance is constructed or generated when the GAMS execution system executes a SOLVE statement. The generated model instance is passed to a solver which searches for a solution of this model instance and returns status information, statistics, and a (primal and dual) solution of the model instance. After the solver terminates, GAMS brings back the solution into the GAMS database, i.e. it updates the level (.L) and marginal (.M) fields of variable and equation symbols used in the model instance. Hence, the SOLVE statement can be interpreted as a complex operator against the GAMS database. The model instance generated by a SOLVE statement only lives during the execution of this one statement, and hence has no representation within the GAMS language. Moreover, its structure does fit the relational data model of GAMS. A model instance consists of vectors of bounds and right hand sides, a sparse matrix representation of the Jacobian, a representation of the non-linear expressions that allow the efficient calculation of gradient vectors and Hessian matrices and so on.

This paper is concerned with solving collections of models that have similar structure but modified data. As an example, consider a linear program of the form:

$$\min c^T x \text{ s.t. } Ax \geq b,\ \ell \leq x \leq u.$$

The data in this problem is (A, b, c, ℓ, u). Omitting some details, the following code could be used within GAMS to solve a collection of such linear programs in which each member of the collection has a different A matrix and lower bound ℓ:

```
Set i / ... /, j / ... /;
Parameter
    A(i,j), b(i);
Variable
    x(j), z, ...;
Equation
    e(i), ...;
e(i).. sum(j, A(i,j)*x(j)) =g= b(i);
...
model mymodel /all/;

Set s / s1*s10 /
```

```
13 Parameter
14     A_s(s,i,j) Scenario data
15     xlo_s(s,j) Scenario lower bound for variable x
16     xl_s(s,j)  Scenario solution for x.l
17     em_s(s,i)  Scenario solution for e.m;
18 Loop(s,
19   A(i,j) = A_s(s,i,j);
20   x.lo(j)= xlo_s(s,j);
21   solve mymodel min z using lp;
22   xl_s(s,j) = x.l(j);
23   em_s(s,i) = e.m(i);
24 );
```

Summarizing, we solve one particular model (`mymodel`) in a loop over `s` with an unchanged model rim (i.e. the same individual variables and equations) but with different model data and different bounds for the variables. The change in model data for a subsequent solve statement does not depend on the previous model solutions in the loop.

The purpose of this new Gather-Update-Solve-Scatter manager or short GUSS is to provide syntax at the GAMS modeling level that makes an instance of a problem and allows the modeler limited access to treat that instance as an object, and to update portions of it iteratively. Specifically, we provide syntax that gives a list of data changes to an instance, and allows these changes to be applied sequentially to the instance (which is then solved without returning to GAMS). Thus, we can simulate a limited set of actions to be applied to the model instance object and retrieve portions of the solution of these changed instances back in the modeling environment.

Such changes can be done to any model type in GAMS, including nonlinear problems and mixed integer models. However, the only changes we allow are to named parameters appearing in the equations and lower and upper bounds used in the model definition.

Thus, in the above example GUSS allows us to replace lines 18-24 by

```
Set dict / s.   scenario. ''
           A.   param.    A_s
           x.   lower.    xlo_s
           x.   level.    xl_s
           e.   marginal. em_s   /;
solve mymodel min z using lp scenario dict;
```

The three dimensional `set dict` (you can freely choose the name of this symbol) contains mapping information between symbols in the model (in the first position) and symbols that supply required update data or store solution information (in the third position), and the type of update/storing (in the second position). An exception to this rule is the tuple with label `scenario` in the second position. This tuple determines the symbol (in the first position) that is used as the scenario index. This scenario symbol can be a multidimensional set. A tuple in this set represents a single scenario. The remaining tuples in the `set dict` can be grouped into input

and output tuples. Input tuples determine the modifications of the model instance prior to solving, while output tuples determine which part of the solution gets saved away. The following keywords can be used in the second position of the set `dict`:

Input:

`param`:	Supplies scenario data for a parameter used in the model
`lower`:	Supplies scenario lower bounds for a variable
`upper`:	Supplies scenario upper bounds for a variable
`fixed`:	Supplies scenario fixed bounds for a variable

Output:

`level`:	Stores the levels of a scenario solution of variable or equation
`marginal`:	Stores the marginals of a scenario solution of variable or equation

Sets in the model cannot be updated. GUSS works as follows: GAMS generates the model instance for the original data. As with regular SOLVE statements, all the model data (e.g. parameter `A`) needs to be defined at this time. The model instance with the original data is also called the *base case*. The solution of the base case is reported back to GAMS in the regular way and is accessible via the regular `.L` and `.M` fields after the SOLVE statement. After solving the base case, the update data for the first scenario is applied to the model. The tuples with `lower`, `upper`, `fixed` update the bounds of the variables, whereas the tuples with `param` update the parameters in the model. The scenario index `k` needs to be the first index in the parameters mapped in the `set dict`. The update of the model parameters goes far beyond updating the coefficients of the constraint matrix/objective function or the right hand side of an equation as one can do with some other systems. GAMS stores with the model instance all the necessary expressions of the constraints, so the change in the constraint matrix coefficient is the result of an expression evaluation. For example, consider a term in the calculation of the cost for shipping a variable amount of goods `x(i,j)` between cities `i` and `j`. The expression for shipping cost is `d(i,j)*f*x(i,j)`, i.e. the distance between the cities times a freight rate `f` times the variable amount of goods. In order to find out the sensitivity of the solution with respect to the freight rate `f`, one can solve the same model with different values for `f`. In a matrix representation of the model one would need to calculate the coefficient of `x(i,j)` which is `d(i,j)*f`, but with GUSS it is sufficient to supply different values for `f` that potentially result in many modified coefficient on the matrix level. The evaluation of the shipping cost term and the communication of the resulting matrix coefficient to the solver are done reliably behind the scenes by GUSS.

After the variable bound and the model parameter updates have been applied and the resulting updates to the model instance data structures (e.g., constraint matrix) has been determined, the modified model instance is passed to the solver. Some solvers (e.g., CPLEX, GUROBI, and XPRESS-OPTIMIZER) allow modifying a model instance. So in such a case, GUSS only communicates the changes from the previous model instance to the solver. This not only reduces the amount of data

communicated to the solver, but also, in the case of an LP model, allows the solver to restart from an advanced basis and its factorization. In the case of an NLP model, this provides initial values. After the solver determines the solution of a model instance, GUSS stores the part of the solution requested by the output tuples of `dict` to some `GAMS` parameters and continues with the next scenario.

3.2.1 GUSS Options

The execution of GUSS can be parameterized using some options. Options are not passed through a solver option file but via another tuple in the `dict` set. The keyword in the second position of this tuple is `opt`. A one dimensional parameter is expected in the first position (or the label `''`). This parameter may contain some of the following labels with values:

`OptfileInit`:	Option file number for the first solve
`Optfile`:	Option file number for subsequent solves
`LogOption`:	Determines amount of log output:
	`0` - Moderate log (default)
	`1` - Minimal log
	`2` - Detailed log
`SkipBaseCase`:	Switch for solving the base case (0 solves the base case)
`UpdateType`:	Scenario update mechanism:
	`0` - Set everything to zero and apply changes (default)
	`1` - Reestablish base case and apply changes
	`2` - Build on top of last scenario and apply changes
`RestartType`:	Determines restart point for the scenarios
	`0` - Restart from last solution (default)
	`1` - Restart from solution of base case
	`2` - Restart from input point

For the example model above the `UpdateType` setting would mean:

```
UpdateType=0:  loop(s, A(i,j) = A_s(s,i,j))
UpdateType=1:  loop(s, A(i,j) = A_base(i,j);
                       A(i,j) $= A_s(s,i,j))
UpdateType=2:  loop(s, A(i,j) $= A_s(s,i,j))
```

The option `SkipBaseCase=1` allows to skip the base case. This means only the scenarios are solved and there is no solution reported back to `GAMS` in the traditional way. The third position in the `opt`-tuple can contain a parameter for storing the scenario solution status information, e.g. model and solve status, or needs to have the label `''`. The labels to store solution status information must be known to `GAMS`, so one needs to declare a set with such labels. The following solution status labels can be reported:

```
domusd    iterusd  objest     nodusd     modelstat  numnopt
numinfes  objval   rescalc    resderiv   resin      resout
resusd    robj     solvestat  suminfes
```

The following example shows how to use some of the GUSS options and the use of a parameter to store some solution status information:

```
Set h solution headers / modelstat, solvestat, objval /;
Parameter
   o / SkipBaseCase 1, UpdateType 1, Optfile 1 /
   r_s(s,h) Solution status report;
Set dict / s.   scenario. ''
           o.   opt.      r_s
           a.   param.    a_s
           x.   lower.    xlo_s
           x.   level.    xl_s
           e.   marginal. em_s   /;
solve mymodel min z using lp scenario dict;
```

3.2.2 Implementation Details

This section describes some technical details that may provide useful insight in case of unexpected behavior.

GUSS changes all model parameters mentioned in the `dict` set to variables. So a linear model can produce some non-linear instructions (e.g. `d(i,j)*f*x(i,j)` becomes a non-linear expression since `f` becomes a variable in the model instance given to GUSS). This also explains why some models compile without complaint, but if the model is used in the context of GUSS, the compile time check of the model will fail because a parameter that is turned into a variable cannot be used that way any more. For example, suppose the model contains a constraint `e(i).. sum(j$A(i,j), ...)`. If `A(i,j)` is a parameter in the regular model, the compiler will not complain, but if `A` becomes a parameter that shows up in the first position of a `param` tuple in the `dict` set, the `GAMS` compiler will turn `A` into a variable and complain that an endogenous variable cannot be used in a `$`-condition.

The sparsity pattern of a model can be greatly effected by GUSS. In a regular model instance `GAMS` will only generate and pass on non-zero matrix elements of a constraint `e(i).. sum(j, A(i,j)*x(j)) ...`, so the sparsity of `A` determines the sparsity of the generated model instance. GUSS allows to use this constraint with different values for `A` hence GUSS cannot exclude any of the pairs `(i,j)` and generate a dense matrix. The user can enforce some sparsity by explicitly restricting the `(i,j)` pairs: `e(i)..sum(ij (i,j), A(i,j)*x(j)) ...`

The actual change of the `GAMS` language required for the implementation of GUSS is minimal. The only true change is the extension of the `SOLVE` statement

with the term `SCENARIO dict`. Existing language elements have been used to store symbol mapping information, options, and model result statistics. Some parts of the GUSS presentation look somewhat unnatural, e.g. since `dict` is a three dimensional `set` the specification the scenario `set` using keyword `scenario` requires a third dummy label `''`. However, this approach gives maximum flexibility for future extension, allows reliable consistency checks at compile and execution time, and allows to delay the commitment for significant and permanent syntax changes of a developing method to handle model instances at a `GAMS` language level.

3.3 Examples

In this section we discuss three examples that benefit from GUSS. Data envelopment analysis models are discussed in Sect. 3.3.1 and a discussion about cross validation models can be found in Sect. 3.3.2. These example describe in detail the steps from a traditional `GAMS` implementation to a GUSS based model. In Sect. 3.3.3 we present the use of GUSS in an implementation of the stochastic dual dynamic program. As many other decomposition algorithms SDDP requires the solution of many closely related mathematical optimization problems. We discuss in detail the savings in running time when using GUSS compared to a traditional `GAMS` implementation and even an implementation based on a native solver interface.

3.3.1 Data Envelopment Analysis

Data Envelopment Analysis (DEA) models have been used extensively in the literature to study a wide variety of applications [10, 11, 14, 16, 20, 24, 26]. The basic (CCR) DEA model is a collection of models indexed by `k` and defined by

$$\begin{array}{lll} \max_{u,v} & u^T Y_{\cdot k} & \text{(indexed objective)} \\ \text{s.t.} & v^T X_{\cdot k} = 1 & \text{(indexed normalizing constraint)} \\ & u^T Y \leq v^T X & \\ & u, v \geq 0 & \end{array}$$

where X, Y are data matrices. The complete GAMS models discussed in the section including the data is available from the GAMS/DEA web page at http://www.gams.com/contrib/gamsdea/dea.htm.

Without GUSS, a model would be defined and solved in a loop over `k`, requiring the model to be generated multiple times with different instances for each value of `k` as shown below.

3.3.1.1 Standard Loop Version of Primal DEA Model

```
$include dea-data.gms
$include dea-primal.gms

set headers / modelstat, solvestat, objval /;
parameter rep(k,headers) solution report summary;
option limrow=0, limcol=0, solprint=silent, lp=gurobi,
       solvelink=%Solvelink.LoadLibrary%;
loop(k,
  slice(j) = data(k,j);
  solve dea using lp max eff;
  rep(k,'modelstat') = dea.modelstat;
  rep(k,'solvestat') = dea.solvestat;
  rep(k,'objval'   ) = dea.objval;
);
display rep;
```

In this setting we loop over the set k and change the data in the objective function and the first constraint of the model explicitly before each solve. We only output a minimal summary of the solution.

GUSS is an alternative (and more efficient) way to define the individual programs and pass them to any underlying GAMS solver. In this way, individual programs are not regenerated, but are instead defined as data modifications of each other. This reduces overall model generation time. Further, previous solutions can be used as starting points in later solves to speed up overall processing time. The specific GAMS code to achieve this is shown below.

3.3.1.2 GUSS Version of Primal DEA Model

```
$include dea-data.gms
$include dea-primal.gms

parameter eff_k(k) 'efficiency report parameter';

set headers report / modelstat, solvestat, objval /;
parameter scenrep(k,headers) solution report summary,


set dict / k      .scenario.''
           slice  .param.   data
           eff    .level.   eff_k
           scopt  .opt.     scenrep /;
```

```
slice(j) = 0; option lp=gurobi;
solve dea using lp max eff scenario dict;
display scenrep, eff_k;
```

In the GUSS version we indicate the collection of models to be solved using the set `dict` defined on lines 10–13. The solve statement on line 16 includes an extra keyword `scenario` that points to this set. The contents of `dict` are directives to GUSS. The first tuple of `dict` determines the set to be used for the scenario (collection) index, in this case `k`. The second tuple of `dict` states that in each scenario `k`, the parameter `slice` is instantiated using a slice of the parameter data. Essentially, this corresponds to the GAMS statement:

```
slice(j) = data(k,j)
```

Note the scenario index `k` must appear as the first index of the parameter data. The third tuple of `dict` allows the modeler to collect information from each solve and store it into a GAMS parameter. Essentially, the third element of `dict` corresponds to the GAMS statement:

```
eff_k(k) = eff.l
```

that gets executed immediately after the solve of scenario `k`. GUSS options (`scopt`) and a parameter to store model statistics (`scenrep`) are given in the last tuple of `dict` indicated by the keyword `opt`.

More complex scenario models can also be formulated using GUSS, including multiple equations being updated. This is shown by the dual of the basic DEA model, given by

$$
\begin{array}{lll}
\min_{z,\lambda} & z & \text{(objective)} \\
\text{s.t.} & X\lambda \leq zX_{\cdot k} & \text{(indexed constraint)} \\
& Y\lambda \geq Y_{\cdot k} & \text{(indexed constraint)} \\
& \lambda \geq 0 &
\end{array}
$$

The original GAMS formulation using standard loops is explicitly given below:

3.3.1.3 Standard Loop Version of Dual DEA Model

```
$include dea-data.gms
$include dea-dual.gms

parameter rep summary report;
option limrow=0, limcol=0, solprint=silent, lp=gurobi
       solvelink=%Solvelink.LoadLibrary%;

loop(k,
   slice(j) = data(k,j);
   solve deadc using lp minimizing z ;
```

```
   rep(k,'modelstat') = deadc.modelstat;
   rep(k,'solvestat') = deadc.modelstat;
   rep(k,'objval') = deadc.objval;
);

display rep;
```

The dual (CRS) DEA model formulated using GUSS is a simple modification, namely:

3.3.1.4 GUSS Version of Dual DEA Model

```
$include dea-data.gms
$include dea-dual.gms

parameter eff_k(k) 'efficiency report parameter';

set headers report / modelstat, solvestat, objval /;
parameter scenrep(k,headers) solution report summary
          scopt / SkipBaseCase 1 /;

set dict / k.     scenario.''
           scopt. opt.     scenrep
           slice. param.   data
           z.     level.   eff_k   /;

slice(j) = 0; option lp=gurobi;
solve deadc using lp min z scenario dict;
display scenrep, eff_k;
```

Because the base model is not solved (due to option `SkipBaseCase 1`), no solution is reported back to GAMS in the traditional way. Solutions for all of the programs can be collected into GAMS parameters as shown above for the `eff_k` parameter.

The aforementioned model is not the only DEA model that exists. Other DEA models that address application issues have been developed and used in practice. Some of these models are simple modifications of the CCR model; other vary more. With the variety of models available, each addressing different needs, GAMS is an important tool to facilitate the definition of general DEA models; GUSS enables fast solution.

Further extensions of these models [1–3, 9, 25] to formulations with weighted outputs or variable returns to scale are easy to formulate with GUSS. One such extended model is also given on the GAMS/DEA web page. It implements the following primal dual pair that incorporates variable returns to scale (VRS) and additive modeling:

$$\begin{aligned} \max_{u,v} \; & u^T Y_{\cdot k} - \mu \\ \text{s.t.} \quad & v^T X_{\cdot k} = 100 \\ & u^T Y \le v^T X + \mu e \\ & u \ge u_{lo}, \; v \ge v_{lo} \end{aligned}$$

and

$$\begin{aligned} \min_{z,\lambda,s,t} \; & 100z - u_{lo}^T s - v_{lo}^T t \\ \text{s.t.} \quad & X\lambda + s = zX_{\cdot k} \\ & Y\lambda - t = Y_{\cdot k} \\ & e^T \lambda = 1 \\ & \lambda, s, t \ge 0 \end{aligned}$$

3.3.2 Cross Validation for Support Vector Machines

Cross validation [13, 17, 19, 23] is a statistical/machine learning technique that aims to evaluate the generalizability of a classifier (or other decision) process. It does this by setting aside a portion of the data for testing, and uses the remaining data entries to produce the classifier. The testing data is subsequently used to evaluate how well the classifier works. Cross validation performs this whole process a number of times in order to estimate the true power of the classifier.

Tenfold cross validation is a special case, where the original data is split into ten pieces, and cross validation is performed using each of these ten pieces as the testing set. Thus, the training process is performed ten times, each of which uses the data obtained by *deleting* the testing set from the whole dataset. We show below how to carry this out using GUSS.

The following example compares the two formulations for a feature-selection model under cross-validation. The complete model and data files are available from the GAMS/Cross Validation web page at http://www.gams.com/contrib/gamsdea/dea-cv.htm.

Original GAMS formulation:

3.3.2.1 Cross Validation Model

```
$include cv-data.gms
$eolcom !

set headers report / modelstat, solvestat, objval /;
parameter rep(p,headers);
option limrow=0, limcol=0, solprint=silent, mip=xpress,
       solvelink=%Solvelink.LoadLibrary%, optcr=0, optca=0;
```

```
$echo loadmipsol=1 > xpress.opt

loop(p,
  a_err.up(a) = inf; a_err.up(a)$a_test(p,a) = 0;
  b_err.up(b) = inf; b_err.fx(b)$b_test(p,b) = 0;
  sla.fx(a) = 0; sla.up(a)$a_test(p,a) = inf;
  slb.fx(b) = 0; slb.up(b)$b_test(p,b) = inf;
  solve train using mip minimizing c;
  train.optfile = 1; ! use mipstart for the second run
  rep(p,'modelstat') = train.modelstat;
  rep(p,'solvestat') = train.solvestat;
  rep(p,'objval') = train.objval;
);
display rep;
```

The `batinclude` file `gentestset.inc` gives instructions for generating the testing sets. It produces `a_test` and `b_test` that detail which equations are left out on solve `p`.

The actual model is set up to include all the data points in the equations `a_def` and `b_def`. To delete the equations that correspond to the test set, we introduce nonnegative slack variables into all the equations. We then set the upper bounds of the slack variables to zero in equations corresponding to the training set, and to infinity in equations corresponding to the testing set. At the same time we fix the error measures `a_err` and `b_err` belonging to the testing set by setting their upper bounds to zero. Thus the testing set equations are always satisfiable by choice of the slack variables alone—essentially they are discarded from the model as required. An alternative formulation could "include" the data equations that you need in each scenario, but the update from one scenario to the next in the defining data is much larger.

Cross validation formulated with GUSS: This model essentially mimics what the standard model does, but the implementation of the solver loop behind the scenes is much more efficient.

3.3.2.2 Cross Validation Using GUSS

```
$include cv-data.gms

parameter wval(p,o), gval(p);

set headers report / modelstat, solvestat, objval /;
parameter
    scenrep(p,headers)
    scopt(*) / SkipBaseCase 1, Optfile 1, LogOption 2 /;

set dict / p.     scenario.''
           scopt. opt.     scenrep
           a_err. upper.   aupper
           b_err. upper.   bupper
           sla.   upper.   afree
           slb.   upper.   bfree
```

```
           weight.level.   wval
           gamma. level.   gval /

$echo loadmipsol=1 > xpress.opt

Parameter aupper(p,a), bupper(p,b), afree(p,a), bfree(p,b);

aupper(p,a)$(not a_test(p,a)) = inf;
bupper(p,b)$(not b_test(p,b)) = inf;

afree(p,a)$a_test(p,a) = inf;
bfree(p,b)$b_test(p,b) = inf;

option mip=xpress, optcr=0, optca=0;
solve train using mip minimizing c scenario dict;
display scenrep, gval;
```

The key observations on this implementation are as follows. Firstly, parameters `aupper`, `bupper`, `afree` and `bfree` are used to set the bounds on the error and slack variables in the testing set equations respectively. The setting of the upper bounds are governed by the syntax shown in the controlling `set dict`. Furthermore, the output of the classifier (`weight`, `gamma`) for each fold of the cross validation uses the `dict`' set to place results into the parameters `wval` and `gval` respectively. Finally, the GUSS options are used to guarantee that the subsequent solves are instructed to process solver options (`Optfile 1`) which instruct the solver to use the previous solution to start the branch-and-cut process (`loadmipsol=1`).

3.3.2.3 Quadratic Programs

GUSS is not limited to linear programs, but can be used more generally. The following example illustrates the use of GUSS for quadratic programs. In this example, a support vector machine is used to determine a linear classifier that separates data into two categories. We use the following model:

$$\begin{array}{ll} \min_{w,g,z} & \frac{1}{2}\|w\|_2^2 + C\sum_i z_i \\ \text{s.t.} & D(Aw - g) + z \geq 1 \\ & z \geq 0 \end{array}$$

Here, A is a matrix containing the training data (patients by features) and D is a diagonal matrix with values $+1$ or -1 (each denoting one of the two classes). C is a parameter weighting the importance of maximizing the margin between the classes ($\frac{2}{\|w\|_2}$) versus minimizing the misclassification error (z). The solution w and g are used to define a separating hyperplane $\{x|w^T x = g\}$ to classify (unseen) data points.

As given, the standard linear support vector machine is not a slice model per se. It becomes a slice model under cross validation training, where it is solved multiple times on different pieces of data. In this case, only the data A and D vary between solves, appropriately fitting the definition of a slice model.

The data for this example comes from the Wisconsin Diagnosis Breast Cancer Database, and is available at http://www.cs.wisc.edu/~olvi/uwmp/cancer.html. The data was converted to the GAMS file `wdbc.gms`, which defines A and D and is also available from the GAMS/Cross Validation web page.

3.3.2.4 The GUSS Formulation for Quadratic SVM

```
$title Ten-fold cross validation example using the scenario solver
$eolcom !

$setglobal num_folds 10

set p folds to perform /1*%num_folds%/;

* Read in data
$include "wdbc.gms"

set test(p,i) 'testing set';

* Define problem
parameter C /1/;
positive variables z(i);
variables obj, w(k), gamma, slack(i);

equations obj_def, sep_def(i);

obj_def..    obj =e= 1/2*sum(k, sqr(w(k))) + C*sum(i, z(i));
sep_def(i).. D(i)*(sum(k, A(i,k)*w(k)) - gamma) + z(i) + slack(i) =g=1;

model train /all/;

* Generate testing sets (to be deleted in each problem)
loop(p,
$batinclude gentestset2.inc "p,i"
);

set h headers / modelstat, solvestat, objval /;
parameter scenrep(p,h), scopt / SkipBaseCase 1 /;
set dict / p.    scenario.''
           scopt.opt.     scenrep
           z.    upper.   iupper
           slack.upper.   ifree /;

Parameter iupper(p,i), ifree(p,i);
iupper(p,i)$(not test(p,i)) = inf;
ifree(p,i)$test(p,i)        = inf;

option qcp=conopt, optcr=0, optca=0;
solve train using qcp minimizing obj scenario dict;
display scenrep;

$if not set runtraditional $exit
```

```
* Traditional Solve
parameter rep(p,h);
option limrow=0, limcol=0, solprint=silent,
       solvelink=%Solvelink.LoadLibrary%;
loop(p,
  z.up(i) = inf; z.up(i)$test(p,i) = 0;
  slack.up(i) = 0; slack.up(i)$test(p,i) = inf;
  solve train using qcp minimizing obj;
  rep(p,'modelstat') = train.modelstat;
  rep(p,'solvestat') = train.solvestat;
  rep(p,'objval') = train.objval;
);
display rep;
```

The variable values for `weight` and `gamma` could be saved for later testing using the same method as detailed above for the linear case.

The batinclude file `gentestset2.inc` is very similar to `gentestset.inc` from the earlier cross-validation examples. In `gentestset2.inc`, though, only one set is being dealt with rather than two.

3.3.3 SDDP

In the last two sections we did not quantify the performance improvements achieved by GUSS. In this section we explore the use of GUSS in a decomposition algorithm applied to a large scale model. We discuss in detail the running times of a traditional GAMS implementation and a GUSS version of the implementation. We also compare the running time of an implementation of the algorithm using the ILOG Concert Technology interface to the CPLEX solver.

The Stochastic Dual Dynamic Programming (SDDP) algorithm [21, 22, 27] for solving multi-stochastic linear programs uses, similar to the well known Benders decomposition [4], the concept of a future cost function (FCF). The algorithm works with an underestimating approximation of this FCF by iteratively adding supporting hyperplanes (Benders cuts) and therefore improving the approximation. Let us consider the following multi-stage stochastic linear program [5]

$$
\begin{aligned}
&\min\ c_1 x_1 + E[\min c_2(\xi_2)x_2(\xi_2) + \cdots + E[\min c_H(\xi_H)x_H(\xi_H)]\cdots] \\
&\text{s.t.}\ \ W_1 x_1 = h_1 \\
&\qquad T_1(\xi_2)x_1 + W_2(\xi_2)x_2(\xi_2) = h_2(\xi_2) \\
&\qquad\quad \vdots \\
&\qquad T_{H-1}(\xi_H)x_{H-1}(\xi_{H-1}) + W_H(\xi_H)x_H(\xi_H) = h_H(\xi_H) \\
&\qquad x_1 \geq 0,\ x_t(\xi_t) \geq 0,\ t = 2, ..., H,
\end{aligned}
$$

where ξ_t are random variables. The SDDP algorithm requires a Markovian structure of the coefficient matrix, meaning that a stage only depends on the previous stage. Furthermore the random variables must be stage-wise independent and must follow a discrete distribution. This means ξ_t can be described as $\xi_t = (\xi_{1t}, ..., \xi_{It})$ for a discrete distribution of I realizations with respective probability p_i.

The SDDP algorithm decomposes the stochastic linear problem into H subproblems of the form

$$\begin{aligned} &\min\ c_t x_t + \hat{\alpha}_{t+1} \\ &\text{s.t.}\ \ W_t x_t \geq h_t - T_{t-1} x^*_{t-1}, \\ &\quad \hat{\alpha}_{t+1} + \pi^j_{t+1} T_t x_t \geq \delta^j_t,\ j = 1, ..., J, \\ &\quad x_t \geq 0, \end{aligned} \qquad (SUB(t, \xi_{it}, x^*_{t-1}))$$

where $\hat{\alpha}_{t+1}$ is represented by free scalar variables. For reason of convenience we omit the random variables in the description. In order to be able to solve a subproblem the previous stage decision variable x^*_{t-1} must be fixed and therefore goes to the right hand side. The set $j = 1, ..., J$ denotes the added hyperplanes to the subproblem, serving as an approximation of the FCF. Throughout the algorithm these kind of subproblems, with different parameters and variable fixings, are the only problems solved. In order to avoid generating and solving them one at a time GUSS allows solving them in certain batches, generating the submodel once for each batch. The details of this process will be shown later in this section.

3.3.3.1 Building of Cuts

Each iteration of the SDDP algorithm consists of two phases: a backward recursion and a forward simulation. In the backward recursion supporting hyperplanes of the form

$$\hat{\alpha}_{t+1} + \pi^j_{t+1} T_t x_t \geq \delta^j_t, \qquad (CUT^j_t)$$

are added to the subproblems in order to improve the approximate FCF. Suppose we are in iteration j and stage $t+1$ of the backward recursion of the algorithm and have solved the subproblem $SUB(t+1, \xi_{i,t+1}, x^*_{t-1})$ for all $i = 1, ..., I$ and dual multipliers $\pi_{i,t+1}$ on all constraints with variables of stage t are stored. In particular this means we may have dual multipliers $\lambda^j_{i,k,t+1}$ belonging to cuts that have been added in earlier iterations. We are now moving to the previous stage t and want to build CUT^j_t. For the coefficient of the cut the sum $\pi^j_{t+1} = \sum_{i=1}^{I} p_i \pi_{i,t+1}$ is calculated. Calculating $\lambda^j_{k,t+1}$ is done in the same way. The cut right hand side δ^j_t is then calculated as follows

$$\delta^j_t = \begin{cases} \sum_{i=1}^{I} p_i \pi^j_{i,t+1} h_{i,t+1}, & t = H-1 \\ \sum_{i=1}^{I} p_i \pi^j_{i,t+1} h_{i,t+1} + \sum_{k=1}^{j} \lambda^j_{k,t+1} \delta^k_{t+1}, & t = 1, ..., H-2. \end{cases}$$

A lower bound is computed by solving the first-stage subproblem with cuts. In the forward simulation the approximate FCF is used to construct a feasible solution of the problem, resulting in an upper bound to the optimal solution value. While going forward we sample one realization out of the set $\{\xi_{1t}, ..., \xi_{It}\}$ and solve the respective subproblem. Note that the only cuts described in this section are

optimality cuts. Usually a second type of cut, feasibility cuts, are used for a Benders decomposition. By adding slack variables we made sure that these are not needed for our model.

3.3.3.2 Algorithm in Pseudo-Code

In the actual algorithm both the backward and forward part is passed through with multiple solutions in one iteration. These solutions are called trial solutions and they are important for several reasons. In the backward recursion this leads to one additional cut per trial solution, which results in a better approximation of the FCF. In the forward simulation the trial solutions help to get a more reasonable estimate of the upper bound. The algorithm in pseudo-code reads as follows

1: **while** convergence is not reached **do**
2: **for** $t = H, ..., 2$ (Backward recursion) **do**
3: **for** each trial solution x^*_{t-1} **do**
4: **for** each realization ξ_{it} of the random variable **do**
5: Solve $SUB(t, \xi_{it}, x^*_{t-1})$ and calculate dual multipliers of the constraints.
6: **end for**
7: **end for**
8: Use dual multipliers to construct the cuts CUT^j_{t-1} and add them to $SUB(t-1, \xi_{i,t-1}, x^*_{t-2})$.
9: **end for**
10: **for** $t = 1, ..., H$ (Forward simulation) **do**
11: **for** each trial solution x^*_{t-1} **do**
12: Solve $SUB(t, \xi_{it}, x^*_{t-1})$ for a sampled realization ξ_{it} and store the solution as x^*_t. Fix x^*_t for $SUB(t+1, \xi_{i,t+1}, x^*_t)$.
13: **if** $t = 1$ **then**
14: store the objective as LOWER BOUND
15: **end if**
16: **end for**
17: **end for**
18: Calculate the UPPER BOUND using the stored solutions.
19: Check for convergence.
20: **end while**

GUSS allows us to rewrite the various `SOLVE` statements in GAMS in the inner loop of the backward recursion (lines 3–7) into one `SOLVE` statement. In specific, we write all possible combinations of trial solutions and realizations into the scenario `dict`. This results in one `SOLVE` statement per stage and each of these `SOLVE` statements will solve (#trials $\times$ #realizations) many models without regenerating them. In the forward simulation we can rewrite the `SOLVE` statements in GAMS in

the inner loop (lines 11-12). This again results in one `SOLVE` statement per stage instead of having #trials many `SOLVE` statements.

3.3.3.3 Results

The SDDP algorithm has been implemented for a stochastic linear program motivated by Vattenfall Energy Trading, a branch of the Swedish power company Vattenfall. The objective of the model is to minimize the power generation costs and ultimately to forecast power prices of the market. Power can be generated by an aggregated hydro power plant, a coal plant, or a nuclear plant. Using hydro power has no costs, but there is a limited amount of water in the reservoir and limited inflows of water into the reservoir over time. In each time period, water can be either used for power generation, saved, or spilled. The model has a granularity of hours and is set up for one year, which results in 8,736 time periods. Stochastic information is revealed every week, resulting in 52 stages. A one-stage sub-model therefore consists of 168 h. In order to compare different implementations of the algorithm, we recorded the time for the first 20 iterations. We used five trial solutions and a discrete distribution made of twelve realizations. During the course of the 20 iterations 66,320 linear programs have been solved. To make runs comparable we worked with a random but fixed sampling (line 12 of the algorithm) in all implementations. All experiments were carried out on a PC with an Intel i7-680 chip running Windows 7 (64bit) with GAMS version 23.7.0 and Cplex 12.2.0.2. We implemented three versions of the algorithm:

Traditional: This is a GAMS model implementing the SDDP algorithm with traditional GAMS programming flow control structures like `loop`. This traditional version has been tested with three different ways of calling the LP solver which is parameterized by the GAMS option `solvelink`. `ChainScript`, which is the default in GAMS, produces for each `SOLVE` statement some scratch files on disk containing the model instance, it also dumps the entire GAMS database into a scratch file and stops the GAMS runtime system leaving all computer resources to the solver job. After the solver job terminates, the GAMS runtime system reinitializes itself from the GAMS database scratch file and continues with execution of the GAMS program. `CallModule` also creates creates files for a model instance but the GAMS runtime system stays in memory while the solver job runs. `LoadLibrary` communicates the model instance through memory and initiates the solver through a shared library. This implementation is available at http://www.gams.com/modlib/adddocs/sddp_trad.gms.

GUSS: This implementation replaces parts of the traditional `loop` constructs in the traditional GAMS model by scenario based `SOLVE` statements using GUSS as discussed above. This model is part of the GAMS Model Library (http://www.gams.com/modlib/libhtml/sddp.htm)

Concert: This implementation is based on ILOG Concert Technology, a programming interface to generate and solve linear, quadratic and constraint programming based models with solvers available from the IBM Cplex Optimization

Studio. This particular C++ implementation used Concert to generate linear programming model and solve them with Cplex. The C++ program is available at http://www.gams.com/modlib/adddocs/sddp.cpp. The model data (and the random data for sampling) comes from GAMS through the Gams Data eXchange (GDX).

All three implementations build on the same core LP technology, the Cplex dual simplex engine. The accumulated time spent in the core Cplex optimizer (`CPXlpopt`) for the 66,320 linear programs amounts to approximately 110 s and differs by less than 1.5% between the different implementations. The following table gives the total running times of the different implementations:

Traditional with `%Solvelink.ChainScript%`	7,204 s
Traditional with `%Solvelink.CallModule%`	2,481 s
Traditional with `%Solvelink.LoadLibrary%`	1,221 s
GUSS	392 s
ILOG **Concert** Technology	210 s

As expected the traditional model becomes faster with a tighter solver communication. The GUSS implementation improves the running time by a factor of more than three compared to the fastest traditional run. The table also shows that the GUSS implementation is slower by a factor of less than two compared to the Concert implementation.

GAMS and other algebraic modeling system are widely accepted as rapid prototyping environments for models and algorithms that require the solution of mathematical optimization problems. Opinions diverge when it comes to selecting the software for deployment of such models and algorithms. One of the most frequent arguments for reimplementing a model in a compiled language with a native solver interface has been the running time overhead of an interpreted language like GAMS. The experiments of this section quantify this overhead (at least for this particular algorithm) and improve the basis for a cost-benefit analysis for the different deployment options.

3.4 Conclusion

While modeling systems such as `GAMS`, `AMPL` and `AIMMS` are often used to prototype and solve optimization problems from large classes of application domains, they have typically been slow at solving collections of models that have simple data changes of a core model. Specific instances that exhibit these difficulties have been described above, ranging from data envelopment analysis, classifier validation, and decomposition approaches such as SDDP, Lagrangian relaxation and Benders approach [4, 8, 12, 15].

We have demonstrated a simple language extension, termed GUSS, that facilitates the gathering of data from different sources/symbols that define the collection of models (that we term scenarios), the procedure that updates a base model instance with this scenario data and then solves this updated model instance, and the mechanism to scatter the scenario results to symbols in the GAMS database. This extension is very easy to incorporate into existing models, and the methodology we use to communicate the information to the solver engine is generalizable to more complex data updating schemes.

The extension can be used with any existing model type within the GAMS environment, and allows data parameters that are present in an existing model to be identified (and updated) using a scenario index. We have demonstrated its utility on a number of example applications and have shown distinct improvements in speed of processing these collections of models. We believe that models updated to use GUSS will be competitive with native implementations of decomposition algorithms, but will have the distinct advantage that they will be much easier to code, available to a modeler to tailor to a specific idea, and enable new suites of problems to be solved directly within the modeling system.

Acknowledgements This work is supported in part by Air Force Grant FA9550-10-1-0101, DOE grant DE-SC0002319, and National Science Foundation Grant CMMI-0928023.

References

1. Banker, R.D., Charnes, A., Cooper, W.W.: Some models for estimating technical and scale inefficiencies in data envelopment analysis. Manag. Sci. **30**(9), 1078–1092 (1984)
2. Banker, R.D., Morey, R.C.: Efficiency analysis for exogenously fixed inputs and outputs. Oper. Res. **34**(4), 513–521 (1986)
3. Banker, R.D., Morey, R.C.: The use of categorical variables in data envelopment analysis. Manag. Sci. **32**(12), 1613–1627 (1986)
4. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numer. Math. **4**(1), 238–252 (1962)
5. Birge, J.R., Louveaux, F.: Introduction to stochastic programming. Springer, London (1997)
6. Bussieck, M.R., Ferris, M.C., Meeraus, A.: Grid enabled optimization with GAMS. INFORMS J. Comput. **21**(3), 349–362 (2009)
7. Bussieck, M.R., Meeraus, A.: General algebraic modeling system (GAMS). In: Kallrath, J. (eds.) Modeling Languages in Mathematical Optimization, pp. 137–157. Kluwer Academic Publishers, Norwell, MA (2003)
8. Carøe, C.C., Schultz, R.: Dual decomposition in stochastic integer programming. Oper. Res. Lett. **24**, 37–45 (1999)
9. Charnes, A., Cooper, W., Lewin, A.Y., Seiford, L.M.: Data envelopment analysis: Theory, Methodology and Applications. Kluwer Academic Publishers, Boston, MA (1994)
10. Charnes, A., Cooper, W.W., Rhodes, E.: Measuring the efficiency of decision making units. Eur. J. Oper. Res. **2**, 429–444 (1978)
11. Cooper, W.W., Seiford, L.M., Tone, K.: Data envelopment analysis: A comprehensive text with models, applications, references and DEA-solver Software. Kluwer Academic Publishers, Boston, MA (2000)
12. Dantzig, G.B., Wolfe, P.: Decomposition principle for linear programs. Oper. Res. **8**, 101–111 (1960)

13. Efron, B., Tibshirani, R.: Improvements on cross-validation: The .632 + bootstrap method. J. Amer. Stat. Assoc. **92**, 548–560 (1997)
14. Farrell, M.J.: The measurement of productive efficiency. J. Roy. Stat. Soc. A (General) **120**(3), 253–290 (1957)
15. Ferris, M.C., Maravelias, C.T., Sundaramoorthy, A.: Simultaneous batching and scheduling using dynamic decomposition on a grid. INFORMS J. Comput. **21**(3), 398–410 (2009)
16. Ferris, M.C., Voelker, M.M.: Slice models in general purpose modeling systems: An application to DEA. Optim. Meth. Software **17**, 1009–1032 (2002)
17. Geisser, S.: Predictive Inference. Chapman and Hall, New York (1993)
18. Kallrath, J. (ed.): Modeling languages in mathematical optimization. Kluwer Academic Publishers, Norwell, MA (2003)
19. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence 2, pp. 1137–1143. Morgan Kaufmann, San Mateo (1995)
20. Olesen, O.B., Petersen, N.C.: A presentation of GAMS for DEA. Comput. Oper. Res. **23**(4), 323–339 (1996)
21. Pereira, M.V.F., Pinto, L.M.V.G.: Stochastic optimization of a multireservoir hydroelectric system: A decomposition approach. Water Resour. Res. **21**(6), 779–792 (1985)
22. Pereira, M.V.F., Pinto, L.M.V.G.: Multi-stage stochastic optimization applied to energy planning. Math. Program. **52**, 359–375 (1991)
23. Picard, R., Cook, D.: Cross-validation of regression models. J. Am. Stat. Assoc. **79**, 575–583 (1984)
24. Seiford, L.M., Zhu, J.: Sensitivity analysis of DEA models for simultaneous changes in all the data. J. Oper. Res. Soc. **49**, 1060–1071 (1998)
25. Simar, L., Wilson, P.W.: Sensitivity analysis of efficiency scores: How to bootstrap in nonparametric frontier models. Manag. Sci. **44**(1), 49–61 (1998)
26. Thanassoulis, E., Boussofiane, A., Dyson, R.G.: Exploring output quality targets in the provision of perinatal care in England using DEA. European J. Oper. Res. **60**, 588–608 (1995)
27. Velaśquez, J., Restrepo, P., Campo, R.: Dual dynamic programming: A note on implementation. Water Resour. Res. **35**(7) (1999)

Chapter 4
Generalized Disjunctive Programming: Solution Strategies

Juan P. Ruiz, Jan-H. Jagla, Ignacio E. Grossmann, Alex Meeraus, and Aldo Vecchietti

Abstract Generalized disjunctive programming (GDP) is an extension of the disjunctive programming paradigm developed by Balas. The GDP formulation involves Boolean and continuous variables that are specified in algebraic constraints, disjunctions and logic propositions, which is an alternative representation to the traditional algebraic mixed-integer programming formulation. GDP has proven to be very useful in representing a wide variety of problems successfully. Even though a wealth of powerful algorithms exist to solve these problems, GDP suffers a lack of mature solver technology. The main goal of this paper is to review the basic concepts and algorithms related to GDP problems and describe how solver technology is being developed. With this in mind after providing a brief review of MINLP optimization, we present an overview of GDP for the case of convex functions emphasizing the quality of continuous relaxations of alternative reformulations that include the big-M and the hull relaxation. We then review disjunctive branch and bound as well as logic-based decomposition methods that circumvent some of the limitations in traditional MINLP optimization. The first implemented GDP solver LogMIP successfully demonstrated that formulating and solving such problems can be done in an algebraic modeling system like `GAMS`. Recently, LogMIP has been introduced into GAMS' Extended Mathematical Programming (EMP) framework

J.P. Ruiz (✉) · I.E. Grossmann
Carnegie Mellon University, Pittsburgh, USA
e-mail: juan.ruiz@visualmesa.com; grossmann@cmu.edu

J.-H. Jagla
GAMS Software GmbH, Braunschweig, Germany
e-mail: jhjagla@gams.com

A. Meeraus
GAMS Development Corporation, Washington D.C., USA
e-mail: ameeraus@gams.com

A. Vecchietti
Universidad Tecnológica Nacional, Santa Fe, USA
e-mail: aldovec@santafe-conicet.gov.ar

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_4, © Springer-Verlag Berlin Heidelberg 2012

integrating it much closer into the GAMS system and language and at the same time offering much more flexibility to the user. Since the model is separated from the reformulation chosen and from the solver used to solve the automatically generated model, this setup allows to easily switch methods at no costs and to benefit from advancing solver technology.

4.1 Introduction

Mixed-integer optimization provides a framework for mathematically modeling many optimization problems that involve discrete and continuous variables. Over the last few years there has been a pronounced increase in the development of these models, particularly in process systems engineering [17,23,28]. Mixed-integer nonlinear programming (MINLP) has made significant progress as a number of codes have been developed over the last decade (e.g. α-ECP [43], BARON [34], Bonmin [6], FilMINT [1], DICOPT [42], SBB [8], LINDOGLOBAL [27]). Despite these advances, the question of how to develop the "best" model that will lead to the most efficient solution method remains to be answered.

Motivated by the above question, one of the trends has been to represent discrete and continuous optimization problems by models consisting of algebraic constraints, logic disjunctions and logic relations [21, 31]. The basic motivation in using these representations is: (a) to facilitate the modeling of discrete and continuous optimization problems, and (b) to retain and exploit the inherent logic structure of problems to reduce the combinatorics and to improve the relaxations.

In this chapter we provide an overview of Generalized Disjunctive Programming [31], which can be regarded as a generalization of disjunctive programming [3]. In contrast to the traditional algebraic mixed-integer programming formulations, the GDP formulation involves Boolean and continuous variables that are specified in algebraic constraints, disjunctions and logic propositions. In the first section of this work we follow the paper by Grossmann and Ruiz [32] to address the solution of GDP problems for the case of convex functions for which we consider the big-M and the hull relaxation MINLP reformulations. We then review disjunctive branch and bound as well as logic-based decomposition methods that circumvent some of the problems encountered in MINLP reformulations. In the second part of this chapter we consider existent solver technologies. We describe the underlying idea of LogMIP and how the integration with EMP leads to a more flexible GDP solver. Finally, we analyze open questions and challenges that solver developers will need to consider when developing new GDP solvers.

4.2 Generalized Disjunctive Programming

The most basic form of an MINLP problem is as follows

$$\begin{aligned} &\min Z = f(x,y) \\ &s.t.\ \ g_j(x,y) \le 0 \quad j \in J \\ &\qquad x \in X \ \ y \in Y \end{aligned} \qquad \text{(MINLP)}$$

where $f : R^n \to R^1, g : R^n \to R^m$ are differentiable functions, J is the index set of constraints, and x and y are the continuous and discrete variables, respectively. In the general case the MINLP problem also involves nonlinear equations, which we omit here for convenience in the presentation. The set X commonly corresponds to a convex compact set, e.g. $X = \{x | x \in R^n, Dx \le d, x^{lo} \le x \le x^{up}\}$; the discrete set Y corresponds to a polyhedral set of integer points, $Y = \{y | y \in Z^m, Ay \le a\}$, which in most applications is restricted to 0-1 values, $y \in \{0,1\}^m$. In most applications of interest the objective and constraint functions $f()$, $g()$ are linear in y (e.g. fixed cost charges and mixed-logic constraints): $f(x,y) = c^T y + r(x)$, $g(x,y) = By + h(x)$. The derivation of most methods for MINLP assumes that the functions f and g are convex [18].

As indicated above, an alternative approach for representing discrete and continuous optimization problems is by using models consisting of algebraic constraints, logic disjunctions and logic propositions [4, 21, 22, 24, 31, 38]. This approach not only facilitates the development of the models by making the formulation process intuitive, but it also keeps in the model the underlying logic structure of the problem that can be exploited to find the solution more efficiently. A particular case of these models is generalized disjunctive programming (GDP) [31], the main focus of this paper. Process Design [17] and Planning and Scheduling [28] are some of the areas where GDP formulations have shown to be successful.

4.2.1 Formulation

The general structure of a GDP can be represented as follows [31]

$$\begin{aligned} &\min\ Z = f(x) + \textstyle\sum_{k \in K} c_k \\ &s.t.\ \ g(x) \le 0 \\ &\qquad \bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ r_{ik}(x) \le 0 \\ c_k = \gamma_{ik} \end{bmatrix} \qquad k \in K \\ &\qquad \Omega(Y) = True \\ &\qquad x^{lo} \le x \le x^{up} \\ &\qquad x \in R^n,\ c_k \in R^1,\ Y_{ik} \in \{True, False\}, i \in D_k, k \in K \end{aligned} \qquad \text{(GDP)}$$

where $f : R^n \rightarrow R^1$ is a function of the continuous variables x in the objective function, $g : R^n \rightarrow R^l$ belongs to the set of global constraints, the disjunctions $k \in K$, are composed of a number of terms $i \in D_k$, that are connected by the OR operator. In each term there is a Boolean variable Y_{ik}, a set of inequalities $r_{ik}(x) \leq 0$, $r_{ik} : R^n \rightarrow R^m$, and a cost variable c_k. If Y_{ik} is True, then $r_{ik}(x) \leq 0$ and $c_k = \gamma_{ik}$ are enforced; otherwise they are ignored. Also, $\Omega(Y) = True$ are logic propositions for the Boolean variables expressed in the conjunctive normal form $\Omega(Y) = \underset{t=1,2,..T}{\wedge} \left[\underset{Y_{ik} \in R_t}{\vee} (Y_{ik}) \underset{Y_{ik} \in Q_t}{\vee} (\neg Y_{ik}) \right]$ where for each clause t, $t = 1, 2 \ldots T$, R_t is the subset of Boolean variables that are non-negated, and Q_t is the subset of Boolean variables that are negated. As indicated in [35], we assume that the logic constraints $\underset{i \in D_k}{\vee} Y_{ik}$ are contained in $\Omega(Y) = True$.

4.2.2 Illustrative Example

The following example illustrates how the GDP framework can be used to model the optimization of a process synthesis problem. Figure 4.1 shows an example of the superstructure of a network we want to optimize. In general, the underlying goal of a classical process synthesis problem consists of selecting the process (i.e. process units and mass flow network) that maximizes the profit when selling a product or set of products considering the cost of raw materials and the cost of the process units.

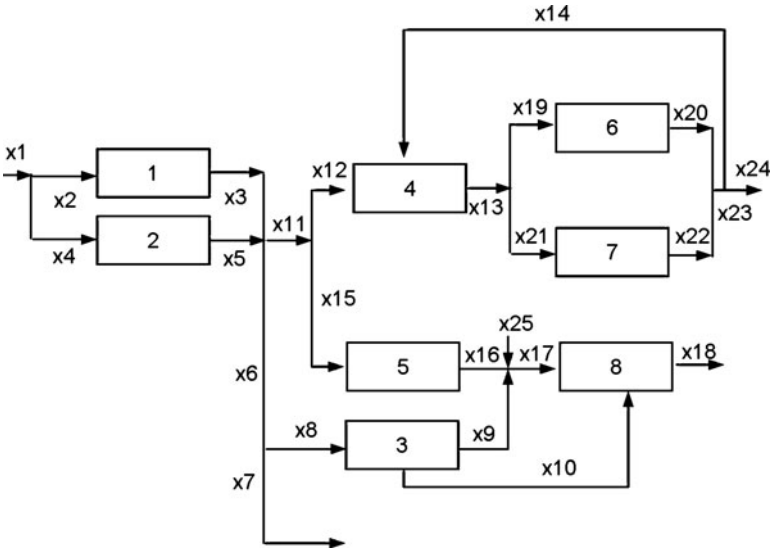


Fig. 4.1 Superstructure for process network

Table 4.1 Nomenclature for process network model

Sets	
J	Set of all process units j
I	Set of all streams i
M	Set of splitter/mixer constraints m
N	Set of constraints restricting mass flows n
Variables	
cf_j	*fixed cost for process* unit j
x_i	Mass flow of stream i
Y_j	Boolean variable associated to process unit j
Parameters	
γ_j	Cost associated to process unit j
d_{ji}	Parameter modifying the exponential term of variable i in unit j
t_{ji}	Parameter modifying the exponential term of variable i in unit j
s_{ji}	Parameter modifying the linear term of the variable i in unit j
r_{in}	Parameter modifying the linear term of the variable i in eq. n
a_{im}	Parameter modifying the linear term of the variable i in eq. m
cv_i	Price/Cost associated to stream i
τ	Fixed rate in objective function

The model in the form of the GDP problem involves disjunctions for the selection of units, and propositional logic for the relationships that exist among these units. Each disjunction contains the equations for each unit (note that these relax as convex inequalities). The nomenclature for this model is shown in Table 4.1.

The objective function of the process synthesis problem can be expressed as follows

$$Min\, Z = \sum_{j \in J} cf_j + \sum_{i \in I} cv_i x_i + \tau$$

Restrictions on flow of materials from one unit to the other arise frequently. For example, pump or pipe characteristics define maximum bounds on the flows. These can be modeled by adding constraints on the flows.

$$\sum_{i \in I} r_{in} x_i \leq 0 \quad \forall n \in N$$

At each mixing/splitting point, a mass balance should be satisfied. This can be modeled as follows.

$$\sum_{i \in I} a_{im} x_i = 0 \quad \forall m \in M$$

The selection or not of each process unit can be modeled by disjunctions. Each disjunction has two terms, the first is associated to the selection of the process unit, in which case, the performance curves of the unit are enforced whereas the second is associated to the no selection of the unit, in which case, all the flows linked to the unit are set to 0.

$$\text{Unit j:} \left[\begin{array}{c} Y_j \\ \sum_i d_{ji}(e^{x_i/t_{ji}} - 1) - \sum_i s_{ji} x_i \leq 0 \\ cf_j = \gamma_j \end{array} \right] \vee \left[\begin{array}{c} \neg Y_j \\ x_i = 0, i \in I^j \\ cf_j = 0 \end{array} \right]$$

Each decision that is taken on each process unit affects the decisions on a subset of the rest of the units. This effect, even though many times implicit in the previous set of equations, can be captured and made explicit through propositional logic by using the logic operations $\wedge$ or $\vee$ and $\Rightarrow$ applied on the boolean variables Y_j. As a result the following GDP is obtained (see Table 4.1 for nomenclature).

$$Min\ Z = \sum_{j \in J} cf_j + \sum_{i \in I} cv_i x_i + \tau$$

$$s.t. \qquad \text{(PROC)}$$

$$\sum_{i \in I} r_{in} x_i \leq 0 \quad \forall n \in N$$

$$\sum_{i \in I} a_{im} x_i = 0 \quad \forall m \in M$$

$$\text{Unit j}: \left[\begin{array}{c} Y_j \\ \sum_i d_{ji}(e^{x_i/t_{ji}} - 1) - \sum_i s_{ji} x_i \leq 0 \\ cf_j = \gamma_j \end{array} \right] \vee \left[\begin{array}{c} \neg Y_j \\ x_i = 0, i \in I^j \\ cf_j = 0 \end{array} \right] \qquad j \in J$$

$$\Omega(Y) = True$$

$$x_i,\ c_j \geq 0, Y_j = \{True, False\}$$

4.2.3 Solution Methods

4.2.3.1 MINLP Reformulation

In order to take advantage of the existing MINLP solvers, GDPs are often reformulated as an MINLP. In order to do so, two main transformations should take place, namely, the disjunctive constraints must be expressed in terms of algebraic equations and the propositional logic needs to be expressed in terms of linear equations.

The disjunctive constraints in (GDP) can be transformed by using either the big-M (BM) [29], or the Hull Relaxation (HR) [24] reformulation.

The former yields:

$$\begin{array}{l} r_{ik}(x) \leq M(1 - y_{ik}) \ \ i \in D_k, \ \ k \in K \\ \sum_{i \in D_k} y_{ik} = 1 \qquad k \in K \\ x \in R^n, y_{ik} \in \{0, 1\}, \ i \in D_k \ , k \in K \end{array} \qquad \text{(DBM)}$$

where the variable y_{ik} has a one-to-one correspondence with the Boolean variable Y_{ik}. Note that when $y_{ik} = 0$ and the parameter M is sufficently large, the associated constraint becomes redundant; otherwise, it is enforced.

The hull reformulation of the disjunction yields

$$
\begin{aligned}
& x = \textstyle\sum_{i \in D_k} \nu^{ik} && k \in K \\
& y_{ik} r_{ik}(\nu^{ik}/y_{ik}) \leq 0 && i \in D_k,\ k \in K \\
& y_{ik} x^{lo} \leq \nu^{ik} \leq y_{ik} x^{up} && i \in D_k,\ k \in K \qquad \text{(DHR)} \\
& \textstyle\sum_{i \in D_k} y_{ik} = 1 && k \in K \\
& x \in R^n,\ \nu^{ik} \in R^n, c_k \in R^1, y_{ik} \in \{0,1\}, i \in D_k,\ k \in K
\end{aligned}
$$

As it can be seen, the HR reformulation is less intuitive than the BM. However, there is also a one-to-one correspondence between disjunctions in (GDP) and (HR). Note that the size of the problem is increased by introducing a new set of disaggregated variables ν^{ik} and new constraints. On the other hand, as proved in Grossmann and Lee [19] and discussed by Vecchietti, Lee and Grossmann [40], the HR formulation is at least as tight and generally tighter than the BM when the discrete domain is relaxed (i.e. $0 \leq y_{ik} \leq 1,\ \ k \in K,\ i \in D_k$). This is of great importance considering that the efficiency of the MINLP solvers heavily rely on the quality of these relaxations.

It is important to note that on the one hand the term $y_{ik}\ r_{ik}\ (\nu^{ik}/y_{ik})$ is convex if $r_{ik}(x)$ is a convex function. On the other hand the term requires the use of a suitable approximation to avoid singularities. Sawaya [35] proposed the following reformulation which yields an exact approximation at $y_{ik} = 0$ and $y_{ik} = 1$ for *any* value of ε in the interval (0,1), and the feasibility and convexity of the approximating problem are maintained.

$$
y_{ik} r_{ik}(\nu^{ik}/y_{ik}) \approx\ ((1-\varepsilon)y_{ik} + \varepsilon) r_{ik}(\nu^{ik}/((1-\varepsilon)y_{ik} + \varepsilon)) - \varepsilon r_{ik}(0)(1 - y_{ik})
$$

Note that this approximation assumes that $r_{ik}(x)$ is defined at $x = 0$ and that the inequality $y_{ik} x^{lo} \leq \nu^{ik} \leq y_{ik} x^{up}$ is enforced.

The transformation of the propositional logic can be accomplished as described in the work by Williams [45] and discussed in the work by Raman and Grossmann [31] and Grossmann et al. [5].

By using the equivalent relations given in the Table 4.2 one can systematically model any arbitrary propositional logic expression that is given in terms of OR, AND and IMPLICATION operators, as a set of linear equality and inequality constraints. One approach is to systematically convert the logical expression into its

Table 4.2 Constraint representation of logic propositions

Logical relation	Boolean expression	Linear inequalities
Logical OR	$P_1 \vee P_2 \vee \ldots P_r$	$y_1 + y_2 + \ldots + y_r \geq 1$
Logical AND	$P_1 \wedge P_2 \wedge \ldots P_r$	$y_1 \geq 1, y_2 \geq 1, \ldots y_r \geq 1$
Implication	$P_1 \Rightarrow P_2$	$1 - y_1 + y_2 \geq 1$

equivalent conjunctive normal form representation, which involves the application of pure logical operations. The conjunctive normal form is a conjunction of clauses $Q_1 \wedge Q_2 \wedge \ldots Q_s$ (i.e. connected by AND operators). Hence, for the conjunctive normal form to be true , each clause must be true independent of the others. Also, since a clause Q_i is a disjunction of literals , $P_1 \vee P_2 \ldots \vee P_s$ (i.e. connected by OR operator) it can be expressed in the linear mathematical form as

$$y_1 + y_2 + \ldots + y_s \geq 1$$

The systematic procedure to transform a logic proposition into an algebraic form consists in first replacing the implication by its equivalent disjunction (e.g. $P_1 \Rightarrow P_2$ by $\neg P_1 \vee P_2$), then move the negation inward by applying DeMorgan's Theorem, (e.g. $\neg(P_1 \wedge P_2)$ is replaced by $\neg P_1 \vee \neg P_2$ and $\neg(P_1 \vee P_2)$ is replaced by $\neg P_1 \wedge \neg P_2$); finally, recursively distribute the "OR" over the "AND" (e.g. $(P_1 \wedge P_2) \vee P_3$ is replaced by $(P_1 \vee P_3) \wedge (P_2 \vee P_3)$).

Having converted each logical proposition into its conjunctive normal form representation $Q_1 \wedge Q_2 \wedge \ldots Q_s$, it can be easily expressed as a set of linear equality and inequality constraints (i.e. $Ay \geq a$ where y represents the binary variables that map the Boolean variables).

As a result we can generate two alternative equivalent MINLP formulations of a GDP, as presented below.

For the case of BM reformulation of the disjunctions

$$\begin{aligned}
&\min\ Z = f(x) + \textstyle\sum_{i \in D_k} \sum_{k \in K} \gamma_{ik} y_{ik} \\
&s.t.\ \ g(x) \leq 0 \\
&\qquad r_{ik}(x) \leq M(1 - y_{ik}) \quad i \in D_k,\ k \in K \qquad \text{(BM)} \\
&\qquad \textstyle\sum_{i \in D_k} y_{ik} = 1 \qquad k \in K \\
&\qquad Ay \geq a \\
&\qquad x \in R^n, y_{ik} \in \{0,1\},\ i \in D_k\ , k \in K
\end{aligned}$$

For the case of HR reformulation of the disjunctions

$$\begin{aligned}
&\min\ Z = f(x) + \textstyle\sum_{i \in D_k} \sum_{k \in K} \gamma_{ik} y_{ik} \\
&s.t.\ \ x = \textstyle\sum_{i \in D_K} v^{ik} \qquad k \in K \\
&\qquad g(x) \leq 0 \\
&\qquad y_{ik} r_{ik}(v^{ik}/y_{ik}) \leq 0 \quad i \in D_k,\ k \in K \qquad \text{(HR)} \\
&\qquad 0 \leq v^{ik} \leq y_{ik} x^{up} \qquad i \in D_k,\ k \in K \\
&\qquad \textstyle\sum_{i \in D_k} y_{ik} = 1 \qquad k \in K \\
&\qquad Ay \geq a \\
&\qquad x \in R^n,\ v^{ik} \in R^n, c_k \in R^1, y_{ik} \in \{0,1\},\ i \in D_k\ , k \in K
\end{aligned}$$

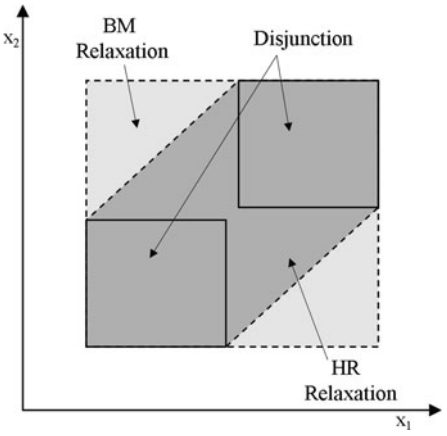


Fig. 4.2 Hull relaxation versus Big-M relaxation

One important question that arises is how to pick between BM and HR. On one hand the relaxation of the HR is generally tighter than BM (see Fig. 4.2 for an illustration) but on the other hand the number of constraints necessary to describe BM is smaller.

These two factors directly affect the efficiency with which the problems will be solved. Even though it is clearly very difficult to state general rules for when to apply one formulation or the other, we can still analyze particular cases. In this work we describe the case in which we have a two terms disjunction with one term defined by the zero point as shown below.

$$\begin{bmatrix} Y \\ r(x) \leq 0 \\ c = \gamma \end{bmatrix} \vee \begin{bmatrix} \neg Y \\ x = 0 \\ c = 0 \end{bmatrix}$$
$$x^{lo} \leq x \leq x^{up}$$

For this case the HR reformulation is given by

$$\begin{aligned} & y r(x/y) \leq 0 \\ & x^{lo} y \leq x \leq x^{up} y \\ & c = \gamma y \\ & y \in \{0, 1\} \end{aligned}$$

whereas the BM reformulation is given by

$$\begin{aligned} & r(x) \leq M(1 - y) \\ & x^{lo} y \leq x \leq x^{up} y \\ & c = \gamma y \\ & y \in \{0, 1\} \end{aligned}$$

Clearly, HR is at least as tight as BM and it uses the same number of constraints. As a result, for this particular case it is often recommended to use the HR reformulation. Note that another interesting question that arises is whether we can generate better formulations by combining the HR and BM reformulations.

Methods that have addressed the solution of problem (MINLP) include the branch and bound method (BB) [7, 20, 25, 37] implemented in MINLP-BB that is based on an SQP algorithm [25] and is available in AMPL, and the code SBB which is available in GAMS [8], Generalized Benders Decomposition (GBD) [16], Outer-Approximation (OA) [10, 13, 46], implemented in DICOPT [42] available in the modeling system GAMS [8], LP/NLP based branch and bound [6,30], and Extended Cutting Plane Method (ECP) [43, 44], implemented in α–ECP and is available in GAMS. An extensive description of these methods and implementations is discussed in the review paper by Grossmann [18] and [32] while some implementation issues are discussed in Liberti et al. [26].

A particular class of GDP problems arises when the functions in the objective and constraints are linear. The general formulation of a linear GDP as described by Raman and Grossmann [31] is as follows.

$$
\begin{aligned}
&\min\ Z = d^T x + \textstyle\sum_k c_k \\
&s.t.\ \ Bx \le b \\
&\quad \bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ A_{ik}x \le a_{ik} \\ c_k = \gamma_{ik} \end{bmatrix} \quad k \in K \qquad\qquad \text{(LGDP)} \\
&\quad \Omega(Y) = True \\
&\quad x^{lo} \le x \le x^{up} \\
&\quad x \in R^n,\ c_k \in R^1,\ \ Y_{ik} \in \{True, False\},\ i \in D_k,\ k \in K
\end{aligned}
$$

The big-M formulation reads

$$
\begin{aligned}
&\min\ Z = d^T x + \textstyle\sum_{i \in D_k} \sum_{k \in K} \gamma_{ij}\, y_{ik} \\
&s.t.\ \ Bx \le b \\
&\quad A_{ik}x \le a_{ik} + M(1 - y_{ik}) \quad i \in D_k, k \in K \quad \text{(LBM)} \\
&\quad \textstyle\sum_{i \in D_k} y_{ik} = 1 \quad k \in K \\
&\quad Ay \ge a \\
&\quad x \in R^n,\ y_{ik} \in \{0, 1\},\ i \in D_k,\ k \in K
\end{aligned}
$$

while the HR formulation reads

$$
\min\ Z = d^T x + \sum_{i \in D_k} \sum_{k \in K} \gamma_{ij} y_{ik}
$$

$$
\begin{aligned}
& s.t.\ x = \sum_{i \in D_K} v^{ik} \quad k \in K \\
& Bx \leq b \\
& A_{ik} v^{ik} \leq a_{ik} y_{ik} \quad i \in D_k,\ k \in K \qquad \text{(LHR)} \\
& y_{ik} x^{lo} \leq v^{ik} \leq y_{ik} x^{up} \quad i \in D_k,\ k \in K \\
& \sum_{i \in D_k} y_{ik} = 1 \quad k \in K \\
& Ay \geq a \\
& x \in R^n,\ v_{ik} \in R^n, c_k \in R^1, y_{ik} \in \{0,1\},\ i \in D_k,\ k \in K
\end{aligned}
$$

Note that in this case the LHR and LBM reformulations are linear mixed-integer programs (MIP). Hence, efficient solvers based on branch and cut algorithms as implemented in CPLEX [9], GUROBI [14], SCIP [36], or XPRESS-OPTIMIZER [12].

4.2.3.2 Logic-Based Methods

In order to fully exploit the logic structure of GDP problems, two other solution methods have been proposed for the case of convex nonlinear GDP, namely, the Disjunctive Branch and Bound method [24], which builds on the concept of Branch and Bound method by Beaumont [4] and the Logic-Based Outer Approximation (LBOA) method [38].

The basic idea in the disjunctive Branch and Bound method is to directly branch on the constraints corresponding to particular terms in the disjunctions, while considering the convex hull of the remaining disjunctions. Although the tightness of the relaxation at each node is comparable with the one obtained when solving the HR reformulation with a MINLP solver, the size of the problems solved are smaller and the numerical robustness is improved.

For the case of Logic-Based Outer-Approximation methods, similar to the case of OA for MINLP, the main idea is to solve iteratively a master problem given by a linear GDP, which will give a lower bound of the solution and an NLP subproblem that will give an upper bound. As described in Turkay and Grossmann [38], for fixed values of the Boolean variables, $Y_{\hat{i}k} = True$, $Y_{ik} = False$ with $\hat{i} \neq i$, the corresponding NLP subproblem (SNLP) is as follows.

$$
\begin{aligned}
& min\ Z = f(x) + \textstyle\sum_{k \in K} c_k \\
& s.t.\ g(x) \leq 0 \\
& \left.\begin{array}{r} r_{\hat{i}k}(x) \leq 0 \\ c_k = \gamma_{\hat{i}k} \end{array}\right\} for\ Y_{\hat{i}k} = True\ \ \hat{i} \in D_k, k \in K \quad \text{(SNLP)} \\
& x^{lo} \leq x \leq x^{up} \\
& x \in R^n\ ,\ c_k \in R^1
\end{aligned}
$$

It is important to note that only the constraints that belong to the active terms in the disjunction (i.e. associated Boolean variable $Y_{ik} = True$) are imposed. This leads to a substantial reduction in the size of the problem compared to the direct application of the traditional OA method on the MINLP reformulation. Assuming that L subproblems are solved in which sets of linearizations $\ell = 1, 2 \ldots . L$ are generated for subsets of disjunction terms $L_{ik} = \{\ell | Y_{ik}^{\ell} = True\}$, one can define the following disjunctive OA master problem (MLGDP).

$$\min \; Z = \alpha + \textstyle\sum_{k \in K} c_k$$

$$s.t.$$

$$\left.\begin{array}{l} \alpha \geq f(x^{\ell}) + \nabla f(x^{\ell})^T (x - x^{\ell}) \\ g(x^{\ell}) + \nabla g(x^{\ell})^T (x - x^{\ell}) \leq 0 \end{array}\right\} \quad \ell = 1, 2 \ldots .., L$$

$$\bigvee_{i \in D_k} \left[\begin{array}{c} Y_{ik} \\ r_{ik}(x^{\ell}) + \nabla r_{ik}(x^{\ell})(x - x^{\ell}) \leq 0 \quad \ell \in L_{ik} \\ c_k = \gamma_{ik} \end{array}\right] k \in K \qquad \text{(MLGDP)}$$

$$\Omega(Y) = True$$

$$x^{lo} \leq x \leq x^{up}$$

$$\alpha \in R^1, \; x \in R^n, \; c_k \in R^1, \; Y_{ik} \in \{True, False\}, \; i \in D_k, \; k \in K$$

It should be noted that before applying the above master problem it is necessary to solve various subproblems (SNLP) for different values of the Boolean variables Y_{ik} so as to produce at least one linear approximation of each of the terms $i \in D_k$ in the disjunctions $k \in K$. As shown by Turkay and Grossmann [38] selecting the smallest number of subproblems amounts to solving a set covering problem, which is of small size and easy to solve. It is important to note that the number of subproblems solved in the initialization is often small since the combinatorial explosion that one might expect is in general limited by the propositional logic. This property frequently arises in Process Networks since they are often modeled by using two terms disjunctions where one of the terms is always linear (see remark below). Moreover, terms in the disjunctions that contain only linear functions need not be considered for generating the subproblems. Also, it should be noted that the master problem can be reformulated as an MIP by using the big-M or Hull reformulation, or else solved directly with a disjunctive branch and bound method.

Remark: In the context of process networks the disjunctions in the (GDP) formulation typically arise for each unit i in the following form.

$$\left[\begin{array}{c} Y_i \\ r_i(x) \leq 0 \\ c_i = \gamma_i \end{array}\right] \vee \left[\begin{array}{c} \neg Y_i \\ B^i x = 0 \\ c_i = 0 \end{array}\right] \quad i \in I$$

in which the inequalities r_i apply and a fixed cost γ_i is incurred if the unit is selected (Y_i); otherwise ($\neg Y_i$) there is no fixed cost and a subset of the x variables is set to zero.

4.3 Generalized Disjunctive Programming Solvers

4.3.1 Logical Mixed Integer Programming (LogMIP)

LogMIP [41] is a software system included in GAMS modeling system for solving problems involving disjunctions and logic propositions. These are formulated as hybrid programs [39] meaning that only the part of the model describing logic disjunctions and propositions is defined using the LogMIP syntax, while the remaining mixed-integer part uses standard features of the GAMS language. In this sense, LogMIP nicely complements the GAMS modeling framework by adding the capability of writing discrete decisions by means of disjunctions. LogMIP includes a language parser interpreting the hybrid program and tools to reformulate these automatically. Linear problems can be reformulated as a mixed-integer program (MIP) using either a linear big-M or hull relaxation [2, 24] respectively. Once the reformulation is performed any MIP solver can be used to solve the problem. One option for solving nonlinear problems is the Logic-Based Outer Approximation [38].

The LogMIP syntax is based on IF .. THEN .. ELSE/ELSIF constructs representing disjunctions of sets of constraints. Each term of a disjunction has to have a Boolean variable mapped to it. The logical operators, implication, equivalence, and negation are used for writing relationships between the Boolean variables in the form of logic propositions. In addition special sentences such as ATLEAST, ATMOST, and EXACTLY, can also used to express relationships for Boolean variables in a more natural and expressive form. More detailed description can be found in www.ceride.gov.ar/logmip.

4.3.2 Extended Mathematical Programming (EMP)

The extended mathematical programming framework EMP was originally developed by M. Ferris and GAMS Development Corp. [11]. EMP was introduced to facilitate communicating higher level structural information existing in a model. The extensible setup of EMP provides the infrastucture for conducting experiments with many non-classical model types and thus prevents the GAMS core language from becoming overloaded. Variables and constraints are joined together using specific optimization and complementarity primitives. The resulting structures, which often involve constraints depending on the solution sets of other models or

complementarity relationships, could be exploited by modern large-scale mathematical programming algorithms to find solutions more efficiently. As an intermediate tool, GAMS introduced the free EMP solver JAMS to reformulate non-traditional problems into established mathematical program classes (e.g. MINLP, NLP), thereby allowing the user to solve these problems by using traditional solver technology. Besides Disjunctive Programs, EMP and JAMS currently support the modeling and solution of Bilevel Programs, Equilibria, Variational Inequalities, Embedded Complementarity Systems, and Extended Nonlinear Programs [15]. Examples are provided in the GAMS EMP Library distributed with each system and available online.[1] In the remainder of this section we will focus on Disjunctive Programs.

The latest developments in Disjunctive Programming solvers are the result of using the powerful technology behind EMP to improve the efficiency and flexibility of LogMIP. By implementing LogMIP as an EMP solver in GAMS we expect to speed up the evolution of LogMIP while making it available to a broader audience. It must be noted that the new EMP-based syntax is very similar to the original LogMIP syntax, so that updating a model to the new syntax is a trivial step.[2] To illustrate this, we present below the formulation of a simple 3-term disjunction using the new syntax.

Disjunction b1 e1 elseif b2 e2 else e3

It is important to note that if the Boolean variables b1 and b2 are not used in other parts of the model it is not necessary to explicitly introduce them into the model. EMP and LogMIP now support the placeholder *. Defining a disjunction using * tells LogMIP to generate the variable internally. The new LogMIP syntax also allows the definition of the reformulation type per disjunction. In the following example the system default[3] is used for the first disjunction, while the big-M with a value of 1e6 becomes the new default, and hence is applied on all subsequent disjunctions. Reformulation through indicator constraints was explicitly assigned to the third disjunction.

*Disjunction * e1 elseif * e2 else e3*
Default bigM 1e6
*Disjunction * e1b elseif * e2b else e3b*
*Disjunction indic * e1c elseif * e2c else e3c*

Table 4.3 shows the new LogMIP syntax including optional statements. More details can be found in the EMP User Manual [15].

[1] http://www.gams.com/emplib/emplib.htm

[2] For comparison, GAMS systems up to 23.6 contain the original LogMIP implementation and are still available for download at http://www.gams.com/download/download_old.htm

[3] ConvexHull, big=1e4, eps=1e-4.

Table 4.3 LogMIP's EMP-based syntax for disjunctions

lDefault	[chull [big eps] | bigM [big eps threshold] | indic]
Disjunction	[chull [big eps] | bigM [big eps threshold] | indic]
	[NOT] var|* equ ELSEIF [NOT] var|* equ [ELSE equ]

Since this new implementation allows the use of indexed and blockwise definitions, the model and LogMIP syntax are easier to read. Another interesting outcome of the integration of LogMIP into EMP is that LogMIP is now available on all platforms where GAMS is available. Finally, the users benefit from EMP features such as the possibility of writing the generated model and the associated dictionary.

In order to handle logic propositions, which often arise in Generalized Disjunctive Programs, GAMS has allowed the use of logical constraints. These new constraints should have particular properties, such as, the equations must not contain operators other than "not", "and", "or", "xor", $\rightarrow$ and $\Leftrightarrow$ and they can only involve binary variables and constants. Below are a few examples for logical propositions within GAMS.

Logic Equation e1; e1.. b1;
Logic Equation e2; e2.. b1 xor b2;
Logic Equation e3; e3.. b1 $\rightarrow$ *not(b2 and b3);*

The advantages of making logic constraints part of the GAMS language are not only to make their specification more natural to GAMS users, but also to allow other solvers to support these types of constraints.

4.3.2.1 Numerical Performance

Among other examples, an instance of the process synthesis problem presented in Sect. 4.2.2 is part of the GAMS Model Library.[4] The implementation makes use of the new LogMIP syntax and GAMS' recently introduced feature for logic constraints. In this section we present the results for an instance of the problem presented. The instance is described by the data below.
Sets
The problem consists of 8 processes (i.e. $J = \{1, 2.., 8\}$) and 25 streams (i.e. $I = \{1, 2 \ldots, 25\}$).

Parameters (The zero parameters are not listed)
For the case of performance equations
$d_{13} = 1$, $d_{25} = 1$, $d_{620} = 1$, $d_{722} = 1$, $d_{818} = 1$, $t_{13} = 1$, $t_{25} = 1.2$, $t_{620} = 1.5$, $t_{722} = 1$, $t_{818} = 1$, $s_{12} = -1$, $s_{24} = -1$, $s_{39} = 1.5$, $s_{38} = -1$, $s_{310} = 1$,

[4] http://www.gams.com/modlib/libhtml/logmip3.htm

$s_{412} = 1.5, s_{414} = 1.5, s_{413} = -1, s_{515} = 1, s_{516} = -2, s_{619} = -1, s_{721} = -1, s_{810} = -1, s_{817} = -1, \alpha_{12} = 1, \alpha_{13} = 1, \alpha_{24} = 1, \alpha_{25} = 1, \alpha_{38} = 1, \alpha_{39} = 1, \alpha_{310} = 1, \alpha_{412} = 1, \alpha_{413} = 1, \alpha_{414} = 1, \alpha_{515} = 1, \alpha_{516} = 1, \alpha_{619} = 1, \alpha_{620} = 1, \alpha_{722} = 1, \alpha_{721} = 1, \alpha_{810} = 1, \alpha_{817} = 1, \alpha_{818} = 1, \gamma_1 = 5, \gamma_2 = 8, \gamma_3 = 6, \gamma_4 = 10, \gamma_5 = 6, \gamma_6 = 7, \gamma_7 = 4, \gamma_8 = 5$

For the objective function
$cv_3 = -10, cv_5 = -15, cv_9 = -40, cv_{19} = 25, cv_{21} = 35, cv_{25} = -35, cv_{17} = 80, cv_{14} = 15, cv_{10} = 15, cv_2 = 1, cv_4 = 1, cv_{18} = -65, cv_{20} = -60, cv_{22} = -80, \tau = 122$

For the splitter/mixer constraints
$a_{13} = 1, a_{15} = 1, a_{16} = -1, a_{111} = -1, a_{213} = 1, a_{219} = -1, a_{221} = -1, a_{317} = 1, a_{39} = -1, a_{316} = -1, a_{325} = -1, a_{311} = 1, a_{312} = -1, a_{315} = -1, a_{46} = 1, a_{47} = -1, a_{48} = -1, a_{523} = 1, a_{520} = -1, a_{522} = -1, a_{523} = 1, a_{514} = -1, a_{524} = -1$

For the flow constraints
$r_{110} = 1, r_{117} = -1.8, r_{210} = 1, r_{217} = -0.4, r_{312} = 1, r_{314} = -5, r_{412} = 1, r_{414} = -2$

The propositional logic used is as follows
$Y_1 \Rightarrow Y_3 \vee Y_4 \vee Y_5; Y_2 \Rightarrow Y_3 \vee Y_4 \vee Y_5$
$Y_3 \Rightarrow Y_1 \vee Y_2; Y_3 \Rightarrow Y_8$
$Y_4 \Rightarrow Y_1 \vee Y_2; Y_4 \Rightarrow Y_6 \vee Y_7$
$Y_5 \Rightarrow Y_1 \vee Y_2; Y_5 \Rightarrow Y_8$
$Y_6 \Rightarrow Y_4; Y_7 \Rightarrow Y_4$
$Y_1 \veebar Y_2; Y_4 \veebar Y_5; Y_6 \veebar Y_7$

In Table 4.4 we compare the performance of two MINLP solvers (i.e. DICOPT and SBB) when using different MINLP reformulation strategies on the (PROC) model. REF 1 represents the pure BM reformulation, namely, each disjunction is replaced by (DBM). REF 4 represents the pure HR reformulation where each disjunction is replaced by (DHR). REF 2 and REF 3 represent two alternative formulations that combine (DBM) and (DHR) constraints. While in the former we use (DBM) constraints for the first seven disjunctions and (DHR) constraints for the eigth one,

Table 4.4 Performance of MINLP solvers using different reformulation strategies

	Optimum	SBB			DICOPT		
		Nodes	LB	Time(sec)	NLP	MIP	Time(sec)
REF 1	68.01	34	–1549.6	2.4	7	7	2.3
REF 2	68.01	24	–1112.5	1.8	5	5	1.6
REF 3	68.01	2	46.8	0.5	2	2	0.8
REF 4	68.01	2	57.1	0.4	2	1	0.5

Table 4.5 Comparison of GDP solution methods

Example	Opt.	Logic based OA			MINLP Reformulation (HR)		
		NLP	MIP	Time(s)	NLP	MIP	Time(s)
8 PROC	68.009	4	1	2.0	2	1	1.4

in the latter we use (DHR) constraints for the first seven disjunctions and (DBM) constraints for the last one.

Clearly, the more (DHR) constraints we add relative to (DBM), the more significant the reduction in the number of nodes and the closer the predicted lower bounds to the global optimum when using SBB. This is a clear indication that (DHR) constraints lead to tighter relaxations. This is also validated by a reduction in the number of NLP and MIP subproblems that are necessary to solve when using DICOPT. Also note that as a result, a benefit in the computational time is observed. However, it is important to realize that sometimes the increase in the size of the formulation given by the (DHR) constraints is not compensated for by the tighter relaxation. In these cases, having the flexibility of replacing only a subset of disjunctions with (DHR) is of great importance.

Finally, Table 4.5 shows the performance of the solver using an MINLP reformulation method and a logic based outer-approximation method. It is important to realize that the size of the NLP subproblems solved by the logic based outer-approximation method is in general much smaller than using the MINLP reformulation. Hence, leading to a more robust performance. This is particular important to consider when dealing with large-scale problems.

4.4 Conclusions

In this chapter, we have presented a review of the basic concepts and algorithms related to GDP problems and described how solver technology is being developed. Solution methods for GDP problems can be divided into two main general strategies. The first one consists of reformulating the GDP problem as a mixed-integer nonlinear program and then solve it using MINLP solvers. This reformulation is not unique and it heavily affects the performance of the solver. With this in mind, in this work we have reviewed reformulation approaches (i.e. HR and BM) that lead to more efficient solution methods. The second strategy consists of exploiting the logic structure of the problems in order to circumvent some of the limitations in traditional MINLP optimization. As a result disjunctive branch and bound as well as logic-based decomposition methods were presented. These two main strategies have been implemented in the GDP solver LogMIP, which works in the GAMS environment. Recently, LogMIP has been introduced into the Extended Mathematical Programming (EMP) framework in GAMS, thus integrating it more tightly into the GAMS system and language, and at the same time offering much

more flexibility to the user. In the last part of this chapter we have described the main capabilities of this solver as well as the syntax to be used when modeling GDP problems. Finally, the performance of the solver has been assessed by solving a process network problem using different techniques. In the last few years many theoretical developments in the area of GDP have been achieved (see [33] and [35]). Currently, the authors of this chapter with GAMS Development Corp. are working on implementing these new techniques to further improve the efficiency when solving problems involving disjunctions and logic propositions.

References

1. Abhishek, K., Leyffer, S., Linderoth, J.T.: FilMINT: An Outer-Approximation-Based Solver for Nonlinear Mixed Integer Programs, ANL/MCS-P1374-0906, Argonne National Laboratory (2006)
2. Balas, E.: Disjunctive programming. **5**, 3–51 (1979)
3. Balas, E.: Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. SIAM J. Alg. Disc. Meth. **6**, 466–486 (1985)
4. Beaumont N.: An algorithm for disjunctive programs. Eur. J. Oper. Res. **48**, 362–371 (1991)
5. Biegler L., Grossmann I.E., Westerberg W.: Systematic methods of chemical process design. Prentice Hall, Englewood Cliffs, NJ, USA (1997)
6. Bonami P., Biegler L.T., Conn A.R., Cornuejols G., Grossmann I.E., Laird C.D., Lee, J., Lodi, A., Margot, F., Sawaya, N., Wächter, A.: An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optim. **5**, 186–204 (2008)
7. Borchers, B., Mitchell, J.E.: An improved branch and bound algorithm for mixed integer nonlinear programming. Comput. Oper. Res. **21**, 359–367 (1994)
8. Brooke A., Kendrick, D., Meeraus, A., Raman R.: GAMS, a User's Guide, GAMS Development Corporation, Washington (1998)
9. www.ibm.com/software/integration/optimization/cplex.
10. Duran, M.A., Grossmann, I.E.: An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Math Program. **36**, 307 (1986)
11. Ferris, M.C., Dirkse, S.P., Jagla, J.-H., Meeraus, A.: An extended mathematical programming framework. Comput. Chem. Eng. **33**, 1973–1982 (2009)
12. www.fico.com
13. Fletcher, R., Leyffer, S.: Solving mixed integer nonlinear programs by outer-approximation. Math Program. **66**, 327 (1994)
14. http://www.gurobi.com
15. GAMS Development Corporation, EMP user's manual, IMA. www.gams.com/solvers/emp.pdf
16. Geoffrion, A.M.: Generalized Benders decomposition, JOTA, **10**, 237–260 (1972)
17. Grossmann, I.E., Caballero, J.A., Yeomans, H. Advances in mathematical programming for automated design, integration and operation of chemical processes. Korean J. Chem. Eng. **16**, 407–426 (1999)
18. Grossmann, I.E.: Review of non-linear mixed integer and disjunctive programming techiques for process systems engineering. Optim. Eng. **3**, 227–252 (2002)
19. Grossmann, I.E., Lee, S.: Generalized convex disjunctive programming: Nonlinear convex hull relaxation. Comput. Optim. Appl. **26**, 83–100 (2003)
20. Gupta, O.K., Ravindran, V.: Branch and bound experiments in convex nonlinear integer programming. Manag. Sci. **31**(12), 1533–1546 (1985)
21. Hooker, J.N., Osorio, M.A.: Mixed logical-linear programming. Discrete Appl. Math. **96–97**, 395–442 (1999)

22. Hooker, J.N.: Logic-based methods for optimization: Combining optimization and constraint satisfaction. Wiley, NY, USA (2000)
23. Kallrath, J.: Mixed integer optimization in the chemical process industry: Experience, potential and future, Trans. I .Chem E. **78**, 809–822 (2000)
24. Lee, S., Grossmann, I.E.: New algorithms for nonlinear generalized disjunctive programming. Comput. Chem. Eng. **24**, 2125–2141 (2000)
25. Leyffer, S.: Integrating SQP and branch and bound for mixed integer nonlinear programming. Comput. Optim. Appl. **18**, 295–309 (2001)
26. Liberti, L., Mladenovic, M., Nannicini, G.: A good recipe for solving MINLPs. Hybridizing metaheuristics and mathematical programming, Springer, **10** (2009)
27. *LindoGLOBAL Solver*, http://www.gams.com/dd/docs/solvers/lindoglobal.pdf
28. Mendez, C.A., Cerda, J., Grossmann, I.E., Harjunkoski I., Fahl, M.: State-of-the-art review of optimization methods for short-term scheduling of batch processes. Comput. Chem. Eng. **30**, 913 (2006)
29. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization, Wiley, New York (1988)
30. Quesada, I., Grossmann, I.E.: An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. Comput. Chem. Eng. **16**, 937–947 (1992)
31. Raman, R., Grossmann, I.E.: Modeling and computational techniques for logic-based integer programming. Comput. Chem. Eng. **18**, 563 (1994)
32. Grossmann, I.E., Ruiz, J.P.: Generalized Disjunctive Programming: a framework for formulation and alternative MINLP optimization. In: Lee, J., Leyffer, S. (Eds.), Mixed Integer Nonlinear Programming. Series: The IMA Volumes in Mathematics and Its Applications, vol. 154, pp. 93–115. Springer, New York (2012)
33. Ruiz, J.P., Grossmann, I.E.: A hierarchy of relaxations for convex generalized disjunctive programs. European Journal of Operational Research **218**, 38–47 (2012)
34. Sahinidis, N.V.: BARON: A general purpose global optimization software package. J. Global Optim. **8**(2), 201–205 (1996)
35. Sawaya, N.: Thesis: Reformulations, relaxations and cutting planes for generalized disjunctive programming. Carnegie Mellon University, Pittsburgh, PA (2006)
36. http://scip.zib.de/
37. Stubbs, R., Mehrotra, S.: A Branch-and-cut method for 0-1 mixed convex programming. Math Program. **86**(3), 515–532 (1999)
38. Turkay, M., Grossmann, I.E.: A Logic-based outer-approximation algorithm for MINLP optimization of process flowsheets. Comput. Chem. Eng. **20**, 959–978 (1996)
39. Vecchietti, A., Grossmann, I.E.: Logmip: A disjunctive 01 non-linear optimizer for process system models. Comput. Chem. Eng. **23**(4), 555–565 (1999)
40. Vecchietti, A., Lee, S., Grossmann, I.E.: Modeling of discrete/continuous optimization problems: Characterization and formulation of disjunctions and their relaxations. Comput. Chem. Eng. **27**, 433–448 (2003)
41. Vecchietti, A., Grossmann, I.E.: *LOGMIP: A discrete continuous nonlinear optimizer*. Comput. Chem. Eng. **23**, 555–565 (2003)
42. Viswanathan and Grossmann I.E.: A combined penalty function and outer-approximation method for MINLP optimization. Comput. Chem. Eng. **14**, 769–782 (1990)
43. Westerlund, T., Pettersson, F.: A Cutting plane method for solving convex MINLP problems. Comput. Chem. Eng. **19**, S131–S136 (1995)
44. Westerlund, T., Pörn, R.: Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. Optim. Eng. **3**, 253–280 (2002)
45. Williams, H.P.: Mathematical building in mathematical programming. Wiley, New York (1985)
46. Yuan, X., Zhang, S., Piboleau, L., Domenech, S.: Une methode d'optimisation nonlineare en variables mixtes pour la conception de procedes. RAIRO, **22**, 331 (1988)

Chapter 5
Xpress–Mosel

Multi-Solver, Multi-Problem, Multi-Model, Multi-Node Modeling and Problem Solving

Susanne Heipcke

Abstract Xpress-Mosel, a commercial product since 2001 (originally developed by Dash Optimization, now FICO), provides a complete environment for developing, testing and deploying optimization applications. Development and analysis of optimization models written with the Mosel language is aided by the graphical environment Xpress-IVE, and tools such as the Mosel debugger and profiler. The Mosel libraries provide the neccessary functionality for a tight integration into existing (C/Java/.NET) applications for model deployment.

This chapter explains the basics of the Mosel language that are required to use the software as a modeling and solution reporting interface to standard matrix-based solvers. It also gives an overview of Mosel's programming functionality. The open, modular design of Mosel makes it possible to extend the Mosel language according to one's needs, adding solvers, data connectors, graphics or system functionality. The second part of this chapter presents possibilities for problem decomposition and concurrent solving from a modeling point of view, with example implementations in Mosel that show handling of multiple models, multiple problems within a model, and as a new feature, distributed computation using a heterogeneous network of computers.

5.1 Introduction

Xpress-Mosel is an environment for modeling and solving problems that is provided either in the form of libraries or as a standalone program. Mosel includes a language that is both a *modeling* and a *programming* language combining the strengths of these two concepts. As opposed to "traditional" modeling environments like AMPL

S. Heipcke (✉)
Xpress Team, FICO, FICO House, International Square, Starley Way, Birmingham, B37 7GN, UK
e-mail: SusanneHeipcke@fico.com; http://www.fico.com/xpress

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_5, © Springer-Verlag Berlin Heidelberg 2012

[7] for which the problem is described using a "modeling language" and algorithmic operations are written with a "scripting language" (similarly for OPL [10] with OPL-script), in Mosel, there is no separation between a modeling statement (e.g., declaring a decision variable or expressing a constraint) and a procedure that actually solves a problem (e.g. call to an optimizing command). Thanks to this synergy, one can program a complex solution algorithm by interlacing modeling and solving statements (cf. [2]).

Each category of problem comes with its own particular types of variables and constraints and a single kind of solver cannot be efficient in all cases. Mosel does not integrate any solver by default but offers a dynamic interface to external solvers provided as *modules*. Each solver module comes with its own set of procedures and functions that directly extends the vocabulary and capabilities of the language. Similar connections are also provided by other systems (cf. [9]) but due to the concept of modules, Mosel is not restricted to any particular type of solver and each solver may provide its specifics at the user level. A major advantage of the modular architecture is that there is no need to modify Mosel's core system to provide access to a new solution technology. The communication between Mosel and its modules uses specific protocols and libraries. This *Native Interface* is public and allows the user to implement his own modules. Any programming task that can be cast into the form of a C program may be turned into a module to be used from the Mosel language.

The Mosel language is a *compiled language*. Distributing models in the form of *platform-independent* compiled models helps to protect intellectual property.
Other components of the Mosel environment are:

- Mosel modules (solvers, data connectors, graphics, system functionality)
- Library interfaces for embedding models into applications (C, Java, C#, VB)
- Mosel Native Interface for the implementation of user modules
- Tools: debugger, profiler
- IVE: visual development environment (Windows 32 and 64-bit)

Since Mosel's first commercialization in 2001, new releases have been published regularly. Table 5.1 summarizes a few highlights of new features that have been introduced—the full list of updates can be found in the release notes of the FICO Xpress distribution.

Contents of this chapter: After a short presentation of Mosel's basic modeling features, this chapter highlights certain advanced features by means of examples. Given the limited space, our description is not systematic and many examples are provided in the form of program extracts with references where to find the complete version of the model code.

Table 5.1 History of major new features in Mosel releases

Version	Release date	Major new features
1.0	November 2001	Product launch in replacement of mp-model (Dash's modeling language since 1983)
1.2	September 2002	Open access to Native Interface; module *mmxslp* (Sequential Linear Programming)
1.4	April 2004	Generalized file handling, I/O drivers; 64bit
1.6	April 2005	Parallelism / multiple models; debugger and profiler; Xpress Application Developer (XAD); Xpress-Kalis (CP solver)
2.0	April 2007	Data structures `list` and `record`; definition of user types; packages (Mosel libraries written with Mosel)
2.2	October 2007	Enhancements to `initializations` functionality (`evaluation of`)
2.4	July 2008	Module *mmnl* (Nonlinear Programming); creation of arrays "on the fly" (`array` operator)
3.0	July 2009	Handling of multiple problems within a single model; *mmxprs:* indicator constraints; *kalis:* automatic linear relaxations
3.2	November 2010	Framework for distributed computing; handling of matrices with more than 2^{31} coefficients

5.2 The Mosel Environment: Quick Tour

5.2.1 Basic Modeling and Solving Tasks

The Mosel language supports all features one might expect of a "classical" algebraic modeling language: problem statement in a close to algebraic form using modeling objects such as decision variables, (linear) constraints, arrays and (index-)sets, easy access to data in external sources, matrix generation and export (in LP or MPS format), and also solver configuration and solution reporting functionality.
The Mosel language has been designed to be easy to learn and to use—a small set of functionality is sufficient for the implementation of standard Mathematical Programming models. You only need to learn about more advanced features if you wish to use them.

5.2.1.1 A First Model

An investor wishes to invest a certain amount of money. He is evaluating ten different securities (the set $SHARES$) for his investment. He estimates the return on investment for a period of one year (RET_s). To spread the risk he wishes to invest at most 30% of the capital into any share. He further wishes to invest at least half of his capital in North-American shares and at most a third in high-risk shares. How should the capital be divided among the shares (with $frac_s$ the fraction invested into shares) to obtain the highest expected return on investment?

The following Mosel model implements and solves this problem as a Linear Programming (LP) problem.

```
model "Portfolio optimization with LP"
 uses "mmxprs"                  ! Use Xpress-Optimizer

 declarations
  SHARES = 1..10                ! Set of shares
  RISK = {2,3,4,9,10}           ! High-risk values among shares
  NA = {1,2,3,4}                ! Shares issued in N.-America
  RET: array(SHARES) of real    ! Estim. return on investment

  frac: array(SHARES) of mpvar  ! Fraction of capital per share
 end-declarations

 RET:= [5,17,26,12,8,9,7,6,31,21]

! Objective: total return
 Return:= sum(s in SHARES) RET(s)*frac(s)

! Limit the percentage of high-risk values
 sum(s in RISK) frac(s) <= 1/3

! Minimum amount of North-American values
 sum(s in NA) frac(s) >= 0.5

! Spend all the capital
 sum(s in SHARES) frac(s) = 1

! Upper bounds on the investment per share
 forall(s in SHARES) frac(s) <= 0.3

! Solve the problem
 maximize(Return)

! Solution printing
 writeln("Total return: ", getobjval)
 forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")
end-model
```

General structure: Every Mosel program starts with the keyword `model`, followed by a model name chosen by the user. The Mosel program is terminated with the keyword `end-model`. All objects must be declared in a `declarations` section, unless they are defined unambiguously through an assignment, e.g., `Return:= sum(s in SHARES) RET(s)*frac(s)` defines `Return` as a linear constraint and assigns to it the expression `sum(s in SHARES) RET(s)*frac(s)`. There may be several such `declarations` sections at different places in a model. In the present case we define three *sets* and two arrays: `SHARES` is a so-called *range set*, i.e. a set of consecutive integers (here: from 1 to 10). `RISK` and `NA` are simply *sets of integers*. `RET` is an array of real values indexed by the set `SHARES`. Its values are assigned after the declarations. `frac`

is an array of decision variables of type `mpvar`, also indexed by the set `SHARES`. These are the decision variables in our model.

The model then defines the objective function, two linear inequality constraints and one equality constraint and sets upper bounds on the variables. Constraints may be named (as is the case here for the objective function `Return`) if they need to be referred to later, for instance, to access solution information (dual, slack, activity).
Solving: With the procedure `maximize`, we call Xpress-Optimizer to maximize the linear expression `Return`. Since there is no "default" solver in Mosel, we specify that Xpress-Optimizer is to be used with the statement `uses "mmxprs"` at the beginning of the program.
Output printing: The last two lines print out the value of the optimal solution and the solution values for all variables.
Line breaks: It is possible to place several statements on a single line, separating them by semicolons (like `RISK = {2,3,4,9,10}; NA = {1,2,3,4}`). Since there are no special "line end" or continuation characters, every line of a statement that continues over several lines must end with an operator (+, >= *etc.*) or characters like `,` that make it obvious that the statement is not terminated.
Comments: As shown in the example, single line comments in Mosel are preceded by `!`. Comments over multiple lines start with `(!` and terminate with `!)`.

Figure 5.1 shows the result of an execution of our model in the development environment Xpress-IVE.

See the "Mosel User Guide" [6] for an in-depth introduction to working with Mosel. The Quick Reference document "MIP formulations and linearizations" [6]

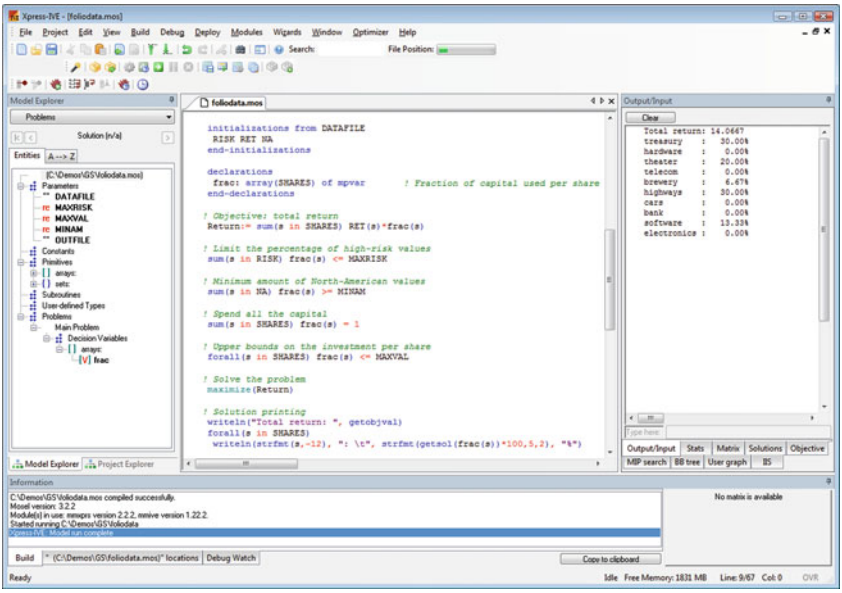


Fig. 5.1 IVE development environment

gives an overview on Mathematical Programming formulation techniques, including the definition of discrete variables, SOS, and indicator constraints with Mosel.

5.2.1.2 Data Handling

Mosel works with a generalized notion of what a "file" is. A "file" may be, for instance:

- A physical file (text or binary)
- A block of memory
- A file descriptor provided by the operating system
- A function (callback)
- A database

The type of the file is indicated by an *extended file name*: the actual file name is preceded by the name of the driver that is to be used to access it. For example, we write `mem:filename` to indicate that we work with a file held in memory, or `mmodbc.odbc:mydata.mdb` to access an MS Access database via the ODBC driver provided by the module *mmodbc*. The default driver (no driver prefix) is the standard Mosel file handling:

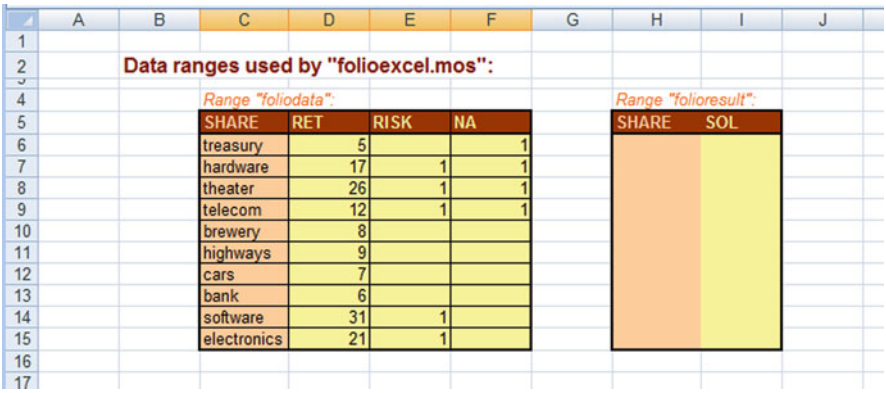
```
declarations
 SHARES: set of string          ! Set of shares
 RISK: set of string            ! High-risk values among shares
 NA: set of string              ! Shares issued in N.-America
 RET: array(SHARES) of real     ! Estim. return on investment
end-declarations

initializations from "folio.dat"
 RISK RET NA
end-initializations
```

This program fragment reads the data file `folio.dat` with the following contents. Note that we are now using sets of type `string` to obtain more meaningful output from the model run. The type change only affects the declaration of the sets. Everything else remains the same.

```
RET: [("treasury") 5 ("hardware") 17 ("theater") 26
      ("telecom") 12 ("brewery") 8 ("highways") 9 ("cars") 7
      ("bank") 6 ("software") 31 ("electronics") 21 ]
RISK: ["hardware" "theater" "telecom" "software" "electronics"]
NA: ["treasury" "hardware" "theater" "telecom"]
```

It is not necessary to initialize the set `SHARES` explicitly. The set is filled when the array `RET`, which is indexed by this set, is read.

Here is an example of how to read the model data from an Excel spreadsheet (using the I/O driver *excel* that is provided by the module *mmodbc*) where all input data is held in a single range labeled "foliodata" as shown in Fig. 5.2.



Data ranges used by "folioexcel.mos":

Range "foliodata":

SHARE	RET	RISK	NA
treasury	5		1
hardware	17	1	1
theater	26	1	1
telecom	12	1	1
brewery	8		
highways	9		
cars	7		
bank	6		
software	31	1	
electronics	21	1	

Range "folioresult":

SHARE	SOL

Fig. 5.2 Model data in excel

```
declarations
  SHARES: set of string            ! Set of shares
  RET: array(SHARES) of real       ! Estim. return on investment
  RISK: array(SHARES) of boolean   ! High-risk values among shares
  NA: array(SHARES) of boolean     ! Shares issued in N.-America
end-declarations

! Data input from spreadsheet
 initializations from "mmodbc.excel:folio.xsl"
   [RET,RISK,NA] as "foliodata"
 end-initializations
```

For further detail on working with spreadsheets and databases please refer to the whitepaper "Using ODBC and other database interfaces with Mosel" [6]. It is also possible to implement one's own I/O drivers—a complete code example for an I/O driver performing compression/decompression of data on the fly is documented in the whitepaper "Generalized file handling in Mosel" [6].

5.2.2 *Programming with Mosel*

Mosel's programming functionality comprises a large set of standard programming language features, including data structures, loops, selection statements, set and list operations, subroutine definition (with overloading), source inclusion and separate compilation of model files.
There is no separation between "modeling statements" and "solving statements". Solution algorithms and even complete interactive graphical application interfaces can be implemented in the same environment as the mathematical models.

5.2.2.1 Structuring Data

Mosel provides the *basic types* that may be expected of any programming language: `integer`, `real` (double precision floating point numbers), `boolean` (symbols `true` and `false`), `string` (single character and any text). Together with the *MP types* `mpvar` (decision variables) and `linctr` (linear constraints) that are provided specifically for mathematical programming, these form the *elementary types* of Mosel.

In addition to the elementary types, Mosel defines the types `set` (collection of elements of a given type without establishing an order among them), `array` (collection of labeled objects of a given type), `list` (ordered collection of elements of a given type which may contain the same element several times), and `record` (finite collection of objects of any type). The operators `+=` and `-=` can be used to add or remove elements from sets and lists. The Mosel language equally supports the set operations union, intersection and difference (operators `+`, `*`, `-`), and a number of specific list handling subroutines (e.g. `reverse`, `getfirst`, `cuthead`).

The following code extract uses a record structure to organize the input data for the portfolio optimization problem:

```
 declarations
  SHARES: set of string           ! Set of shares
  SDATA: array(SHARES) of record  ! Data relating to shares:
    Risk: boolean                 !   High-risk flag
    NA: boolean                   !   Issued in N.-America flag
    Return: real                  !   Est. return on investment
  end-record
 end-declarations

 ...

! Objective: total return
 Return:= sum(s in SHARES) SDATA(s).Return*frac(s)

! Limit the percentage of high-risk values
 sum(s in SHARES | SDATA(s).Risk) frac(s) <= MAXRISK
```

5.2.2.2 Selections and Loops

The simplest form of a *selection* is the `if-then` statement which may be extended to `if-then-else` or even `if-then-elif-then-else` to test two conditions consecutively and execute an alternative if both fail. If several mutually exclusive conditions are tested, the `case` statement should preferably be used.

Loops regroup actions that need to be repeated a certain number of times, either for all values of some index or counter (`forall`) or depending on whether a condition is fulfilled or not (`while`, `repeat-until`). The `forall` and `while`

loops in Mosel exist in two versions: an inline version for looping over a single statement (as we have already see in the initial example) and a second version `forall-do` (`while-do`), that may enclose a block of statements, the end of which is marked by `end-do`.

In order to study the effect of different risk limit values on the solution of our portfolio optmization problem we can introduce a loop around the definition of this constraint and the solution procedure. As problem definition in Mosel is incremental, we need to override the previous constraint definition at every iteration. We therefore now give this constraint a name, `Risk`. If the constraint did not have a name, then each time the loop was executed, a new constraint would be added, and the existing constraint would not be replaced.

```
forall(r in 0..20) do
  ! Redefine limit on the percentage of high-risk values
  Risk:= sum(s in RISK) frac(s) <= r/20
  maximize(Return)                  ! Solve the problem
  if (getprobstat = XPRS_OPT) then  ! Save the solution value
   writeln(r/20, "%: ", getobjval)
  else
   writeln("No solution for high-risk <= ", 100*r/20, "%")
  end-if
end-do
```

5.2.2.3 Subroutines

Mosel provides a set of predefined subroutines (e.g. procedures like `write` / `writeln`, arithmetical functions like `cos`, `exp`, `ln`), but it is also possible to define new procedures and functions according to the needs of a specific program. User-defined subroutines in Mosel have to be marked with `procedure` / `end-procedure` and `function` / `end-function` respectively. The return value of a function has to be assigned to `returned`. It is possible to pass parameters into a subroutine. The (list of) parameter(s) is added in parentheses following the name of the subroutine.

The structure of subroutines is very similar to the one of `model`: they may include `declarations` sections for declaring local parameters that are only valid in the corresponding subroutine. Subroutine calls may be nested, and they may also be called recursively. It is possible to re-use the names of subroutines, provided that every version has a different number and/or types of parameters. This functionality is commonly referred to as *overloading*.

Subroutines defined in a Mosel model can also serve as a means for interacting with external software. Mosel modules such as *mmxprs* allow the user to define *callback functions* in Mosel that are invoked by the external software at predefined places (such subroutines have a predefined signature and need to be marked as `public`). The following example defines a function for printing out the current solution that is called whenever an integer solution is found by Xpress-Optimizer.

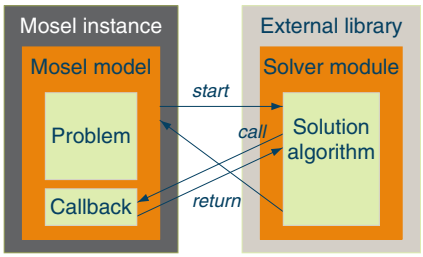


Fig. 5.3 Mosel model with solver callback function

```
! Setting the MIP solution callback
 setcallback(XPRS_CB_INTSOL, "printsol")

! Solve the problem
 maximize(Return)

! Solution printing
 public procedure printsol
  writeln("Total return: ", getsol(Return))
  forall(s in SHARES | getsol(frac(s))>0)
   writeln(s, ": ", getsol(frac(s))*100, "%")
 end-procedure
```

Figure 5.3 depicts the control flow for a model defining a callback function for one of the modules used by it.

Note: If we are interested in finding a large number of alternative, near-optimal solutions we can configure the MIP solver to use the solution enumerator by adding an optional first argument to the optimization call: `maximize(XPRS_ENUM, Return)`. This setting changes the solver behaviour to search larger parts of the branch-and-bound tree, in general implying longer solve times. Figure 5.4 shows the solutions found by standard MIP search and by the solution enumerator for the MIP version of our portfolio example; a code example is given in the appendix to this chapter.

5.2.3 Modules and Packages

An original feature of Mosel is the possibility to extend the language by means of modules (dynamic libraries written in C) and packages (libraries written in the Mosel language). The Mosel Native Interface is public and allows the user to implement his own modules that are treated just like the modules of the Xpress distribution.

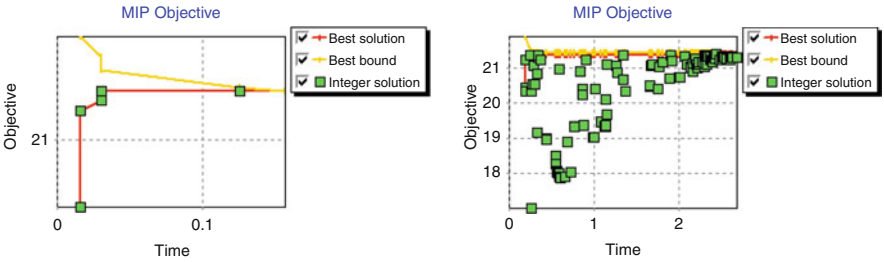


Fig. 5.4 Comparing MIP solutions found with standard MIP search and by the solution enumerator

The Mosel language is open to extensions through user-written libraries. These libraries may take two forms:

- *Package*—a library written in the Mosel language defining new constants, subroutines and types for the Mosel language
- *Module*—a dynamic library (Dynamic Shared Object, DSO) written in the C programming language that observes the conventions set out by the Mosel Native Interface

The structure of *packages* is similar to models, replacing the keyword `model` by `package`. Packages are included into models with the `uses` statement, in the same way as for modules. Unlike Mosel code that is included into a model with the `include` statement, packages are compiled separately, that is, their contents are not visible to the user. Typical uses of packages include:

- Development of your personal "tool box"
- Model parts (e.g., reformulations) or algorithms written in Mosel that you wish to distribute without disclosing their contents
- Add-ons to modules that are more easily written in the Mosel language

A *module* may extend the Mosel language with new

- Constant symbols
- Subroutines
- Types
- I/O drivers
- Control parameters

In this list, subroutines and types are certainly the most important items. *Subroutines* defined by a module may be entirely new functions or procedures or overload existing subroutines of Mosel. A module may, for instance, provide a subroutine that calls an algorithm or a solution heuristic that is readily available in the form of a C or C++ library function.

New *types* defined by a module are treated exactly like the own types of Mosel (like `integer` or `mpvar`). They can be used in complex data structures (arrays, sets *etc.*), read in from file in `initializations` sections, or appear as

parameters of subroutines. The operators in Mosel can be overloaded to work with new types. The definition of new types may be required to support solvers such as finite domain constraint solvers.

The Mosel distribution comes with a set of *I/O drivers* that provide interfaces to specific data sources (such as ODBC) or serve to exchange information between the application running the Mosel libraries and a Mosel model in a very direct way by providing various possibilities of passing data back and forth in memory. The user may define additional drivers, for instance to read/write compressed or encrypted files.

Constants and *control parameters* published by a module make little sense on their own. They will typically be used in conjunction with the module's types or subroutines.

5.2.3.1 Available Modules

At present, the following modules have been implemented:

Solvers: mmxprs, mmquad, mmnl, mmxslp, kalis

Data handling: mmodbc, mmoci, mmetc

System: mmsystem

Model handling: mmjobs

Graphics: mmive, mmxad

In the preceding examples the module *mmxprs* has already been used to solve problems with Xpress-Optimizer. With the help of the module *mmquad* it is possible to formulate and solve Quadratic Programming problems. The module *mmnl* for handling non-linear constraints is not a solver on its own. In combination with *mmxprs* you can formulate and solve Quadratically Constrained Quadratic Programming problems and Linearly Constrained Convex Optimization problems.

General non-linear problems can be formulated and solved with the Successive Linear Programming module *mmxslp*.

The module *kalis* gives access to the Constraint Programming solver Kalis by Artelys.

Modules may also define additional interfaces to data files: *mmetc* defines the procedure `diskdata` that emulates the data in- and output of mp-model [1]; *mmoci* defines a specific interface to Oracle databases; *mmodbc* provides access to any data source for which an ODBC interface is available, using Mosel's `initializations` blocks or, for larger flexibility, through standard SQL commands. This module also defines a software-specific interface to Excel.

With the help of the module *mmive* the user can create his own *graphics* in the graphical environment Xpress-IVE. Taking this even further, the module *mmxad* (Xpress Application Developer, XAD) enables the user to define a complete graphical application from within a Mosel model.

5.2.3.2 User Modules

In addition to the modules provided, Mosel is open to any kind of addition by its users. The communication between Mosel and its modules uses specific protocols and libraries. This *Native Interface* is public and allows the user to implement his own modules (see [4]). Any programming task that can be cast into the form of a C program may be turned into a module to be used from the Mosel language. Possible uses of user-written modules include, but are not limited to:

- Application specific data handling (e.g. definition of composite data types, data input and output in memory)
- Access to external programs,
- Access from the model to solution algorithms and heuristics implemented in a programming language (possibly re-using existing code)
- Access to efficient implementations of specific, time-critical programming tasks (such as a sorting algorithm that needs to be called frequently by a heuristic)

5.2.4 Embedding and Deployment

Mosel models can be embedded into host applications via the Mosel Libraries (available for C, Java, C#, VB). Communication between the host application and a Mosel model typically relies on data exchange in memory. Alternatively, a XAD GUI can be developed directly in the Mosel language.

During its development, a model is typically run as a stand-alone application, either directly from the Mosel command line or in the IVE graphical interface. For its deployment however, a comfortable way of embedding a model into a programming language environment is required. To do this, Mosel provides two libraries; the *Mosel Compiler Library* and the *Mosel Run-Time Libary*. The former transforms models into BIM files (portable **Bi**nary **M**odel files). The latter is required for loading and running these files and subsequently accessing the model information.

The end user of an optimization application does not usually access the model source directly [3]: Mosel models can be distributed as BIM files, thus protecting your intellectual property and also preventing accidental changes for easier maintenance. A (binary) model file contains the problem statement and the definition of solution algorithms, but usually not the problem data. At every model execution, data for a particular problem instance is read from specified external sources. The same configurable Mosel model might be used with different interfaces depending on the type of use (e.g. development with IVE, XAD GUI for end users, Java application for processing large batches of model instances), as shown in Fig. 5.5.

In our portfolio model, we now define the data file name and certain numerical constants are defined as *model parameters*. These model parameters may be reset

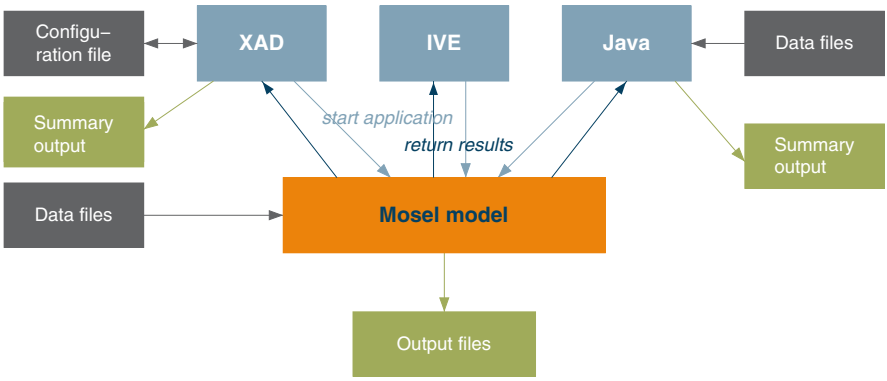


Fig. 5.5 Interface choices for a Mosel model

at the execution of the model or otherwise take their default value given in the `parameters` section.

```
parameters
 DATAFILE= "folio.dat"    ! File with problem data
 MAXRISK = 1/3            ! Max. investment into high-risk
 MAXVAL = 0.3             ! Max. investment per share
 MINAM = 0.5              ! Min. investment into N.-American
end-parameters
```

The following simple Java program changes the name of the data file and the two parameters `MAXRISK` and `MAXVAL` before running the Mosel model (saved with the name `foliodata.mos`). After the optimization run we retrieve the problem status and if a solution was found, print it out. It would also be possible to obtain the full information about the current values of all model objects, including data, decision variables and constraints.

```
import com.dashoptimization.*;
public class runfolio
{
 public static void main(String[] args) throws Exception
 {
  XPRM mosel;
  XPRMModel mod;

  mosel = new XPRM();                          // Initialize Mosel
  mosel.compile("foliodata.mos");               // Compile model
  mod = mosel.loadModel("foliodata.bim");      // Load model
                                  // Set the run-time parameters
  mod.execParams="DATAFILE=data2.dat,MAXRISK=0.4,MAXVAL=0.25";
  mod.run();                                    // Run the model

// Test whether a solution is found and print objective value
  if(mod.getProblemStatus()==XPRMModel.PB_OPTIMAL)
    System.out.println("Solution: " + mod.getObjectiveValue());
 }
}
```

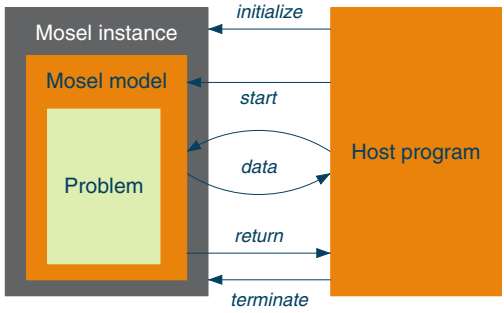
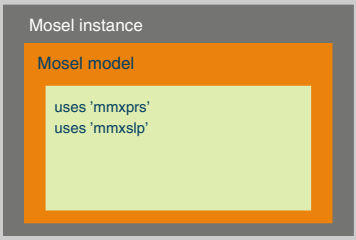


Fig. 5.6 Embedding a Mosel model into a host application

Figure 5.6 shows the typical program control flow when a Mosel model is embedded in a host application: Mosel is initialized from the application that then loads and starts running a particular model. During its execution the model might exchange data (in memory or via physical files) with the calling application and also with other external data sources as discussed in Sect. 5.2.1.2. After termination of the model the control returns to the host application.

5.3 The "Multis"

5.3.1 Multi-Solver



Xpress-Mosel is designed with an open, modular architecture. Solvers are modules that are loaded into a model as needed. The Xpress suite comes with a comprehensive set of optimization solvers from which you can choose the one that is best suited for your problem type, or you can use several solvers in combination within a single model. Other modules—for example providing data interfaces or graphics—are also available and users can even write their own modules to enhance the Mosel language according to their needs.

In the previous section we saw a series of examples solving Mathematical Programming problems with Xpress-Optimizer. However, the Mosel language is not

limited to one particular type of solver. Other solvers in the Xpress distribution are Xpress-SLP for solving nonlinear problems by Sequential Linear Programming, and Xpress-Kalis for Constraint Programming (CP) with finite domain and floating point variables.

Constraint Programming is a different paradigm for representing and formulating optimization problems. In this section we show how to implement a simple job-shop scheduling problem with Xpress-Kalis.

The job-shop problem consists of scheduling NJ jobs on NM machines of capacity 1 (one job at a time). Each job needs to be processed by every machine. The sequence of production tasks and their respective duration on the machines is different for each job. The objective is to complete all jobs as early as possible, that is, to minimize the completion time of the last job (the so-called makespan*).*

With fixed resource assignment RES_{jt} and duration DUR_{jt} the only decision variables in this problem are the start times $start_{jt}$ for all jobs j and tasks t. The condition "one job at a time" is expressed by a global *disjunctive* constraint that guarantees that tasks processed on the same machine do not overlap. The statement of a CP problem always includes the definition of a search strategy. Here we first fix the disjunctions between tasks (whether A comes before B or B before A) and secondly, enumerate the values for the decision variables (start times).

```
model "Job shop (CP)"
 uses "kalis"

 parameters
  NJ = 6                              ! Number of jobs
  NM = 6                              ! Number of resources
 end-parameters

 declarations
  JOBS = 1..NJ                        ! Set of jobs
  TASKS = 1..NM                       ! Set of tasks/resources
  RES: array(JOBS,TASKS) of integer   ! Resource use of tasks
  DUR: array(JOBS,TASKS) of integer   ! Durations of tasks
 end-declarations

 initializations from "jobshop.dat"
  RES DUR
 end-initializations
                              ! Upper bound on total duration
 HORIZON:= sum(j in JOBS, t in TASKS) DUR(j,t)
 setparam("DEFAULT_LB", 0)
 setparam("DEFAULT_UB", HORIZON)

 declarations
  start: array(JOBS,TASKS) of cpvar   ! Start times of tasks
  makespan: cpvar                     ! Makespan
  DURv: array(cpvar) of integer       ! Dur. indexed by var.
  Disj: set of cpctr                  ! Disjunction constraints
  Strategy: array(range) of cpbranching  ! Branching strategy
  sSet: array(TASKS) of set of cpvar
 end-declarations
```

```
! Precedence constr. betw. last task of every job and makespan
 forall (j in JOBS) makespan >= start(j,NM) + DUR(j,NM)

! Precedence constraints between the tasks of every job
 forall (j in JOBS,t in TASKS | t<NM)
  start(j,t) + DUR(j,t) <= start(j,t+1)

! Disjunctions between tasks on the same machine
 forall(j in JOBS,t in TASKS) DURv(start(j,t)):=DUR(j,t)
 forall (r in TASKS) do
  sSet(r):=union(j in JOBS,t in TASKS|RES(j,t)=r) {start(j,t)}
  disjunctive(sSet(r), array(v in sSet(r)) DURv(v), Disj, 0)
 end-do

! Branching strategy
 Strategy(0):= settle_disjunction(Disj)
 Strategy(1):= split_domain(KALIS_LARGEST_MAX,KALIS_MIN_TO_MAX)
 cp_set_branching(Strategy)

! Solve the problem
 if not(cp_minimize(makespan)) then
  writeln("Problem is infeasible.")
 else
  writeln("Total completion time: ", getsol(makespan))
 end-if

end-model
```

An alternative, usually more efficient way of formulating scheduling problems with Xpress-Kalis uses the high-level modeling objects *task* and *resource* and a dedicated scheduling solver. The object *task* contains information relating to a process step. As a minimum, these are its start, duration and completion times, optionally also sets of predecessors or successors, sequence-dependent setup duration, and resource usage or production for renewable or non-renewable resources. A *resource* is characterized by its capacity (unary or cumulative). Optionally, periods of unavailability may be specified and it is also possible to configure the propagation algorithm associated with a resource.

Below is a formulation of the job-shop problem with *task* and *resource* objects. We are now using the scheduling solver with the built-in default scheduling search strategy—it is equally possible to redefine this strategy for particular cases.

```
model "Job shop (tasks)"
 uses "kalis"

 parameters
  NJ = 6                              ! Number of jobs
  NM = 6                              ! Number of resources
 end-parameters
```

```
 declarations
  JOBS = 1..NJ                          ! Set of jobs
  TASKS = 1..NM                         ! Set of tasks/resources
  RES: array(JOBS,TASKS) of integer     ! Resource use of tasks
  DUR: array(JOBS,TASKS) of integer     ! Durations of tasks
  res: array(TASKS) of cpresource       ! Resources
  task: array(JOBS,TASKS) of cptask     ! Tasks
 end-declarations

 initializations from "jobshop.dat"
  RES DUR
 end-initializations

 HORIZON:= sum(j in JOBS, t in TASKS) DUR(j,t)
 forall(j in JOBS) getend(task(j,NM)) <= HORIZON

! Setting up the resources (capacity 1)
 forall(r in TASKS)
  set_resource_attributes(res(r), KALIS_UNARY_RESOURCE)

! Setting up the tasks (durations, resource used)
 forall(j in JOBS, t in TASKS)
  set_task_attributes(task(j,t), DUR(j,t), res(RES(j,t)))

! Precedence constraints between the tasks of every job
 forall (j in JOBS, t in 1..NM-1)
  setsuccessors(task(j,t), {task(j,t+1)})

! Solve the problem
 if cp_schedule(getmakespan)=0 then
  writeln("Problem is infeasible")
 else
  writeln("Total completion time: ", getsol(getmakespan))
 end-if

end-model
```

When using IVE to run this model, the GUI automatically generates a Gantt chart of the best solution (Fig. 5.7).

Xpress-Optimizer and Xpress-Kalis can be used jointly in a Mosel model. Several examples are described in the whitepaper "Hybrid MIP/CP solving with Xpress-Optimizer and Xpress-Kalis" [6]. The Kalis solver can also be run in *automatic linear relaxation* mode, whereby the constraint solver triggers LP or MIP solving with Xpress-Optimizer of automatically generated, configurable subproblems to guide the CP search or improve bound estimates on the objective function. For further detail on this topic and Constraint Programming with Mosel in general the reader is refered to the "Xpress-Kalis User Guide" [6].

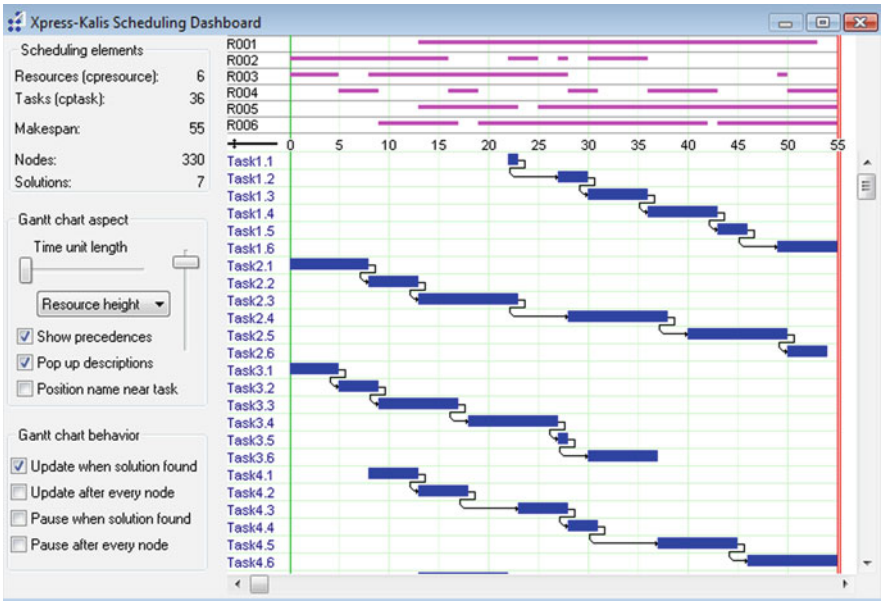
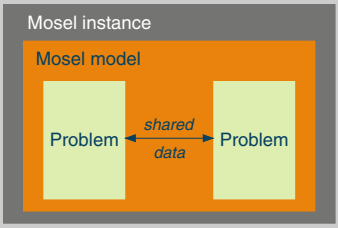


Fig. 5.7 Gantt chart of the solution to the job-shop problem generated by IVE

5.3.2 *Multi-Problem*



With Xpress-Mosel, multiple optimization problems can be defined within a single optimization model. At any point a single problem is active, it is possible to switch back and forth between various other problems, allowing for the retrieval of solution information across problem components. Problems can share data, make use of common decision variables, and copy constraints from one problem to another or duplicate a problem altogether.

Mosel can handle several *problems* in a given *model* file. The type `mpproblem` is used to identify mathematical programming problems. Modules can define other specific problem types and new problem types can also be defined by combining existing ones. A default problem is associated with every model—if you simply want to work with a single problem you don't need to worry about problems.

The statement `with` allows a problem to be opened (= select the active problem). The same decision variable (type `mpvar`) may be used in several problems, taking different solution values in each. Constraints belong to the problem where they are defined.

In terms of an example, let us take a look at how to implement a column generation algorithm for a cutting stock problem.

A paper mill produces rolls of paper of a fixed width MAXWIDTH that are subsequently cut into smaller rolls according to the customer orders. The rolls can be cut into NWIDTHS different sizes. The orders are given as demands for each width i (DEMAND$_i$). The objective of the paper mill is to satisfy the demand with the smallest possible number of paper rolls in order to minimize the losses.

The objective of minimizing the total number of rolls can be expressed as choosing the best set of cutting patterns for the current set of demands. Since it may not be obvious how to calculate all possible cutting patterns by hand, we start off with a basic set of patterns that consists of cutting small rolls all of the same width as many times as possible (and at most the demanded quantity) out of the large roll. After (re)solving this problem a column generation algorithm iteratively adds a new pattern (=column) that improves the current solution. The new columns must have a negative reduced cost in a minimization problem and are calculated based on the dual value of the current solution. The column generation loop stops when no new column improving the current solution can be found.

The column generation subproblem for generating new cutting patterns takes the form of an integer knapsack problem: $\{\max \sum_j C_j x_j : \sum_j A_j x_j \leq B, x_j \text{ integer}\}$, where coefficients C_j are the dual values of the demand constraints, A_j the demanded widths and B is the value *MAXWIDTH*.

The following model extract shows the implementation of a function `knapsack` that can be called repeatedly from the column generation loop. For efficiency reasons we have declared the knapsack variables and constraints globally so as to create them once only. Their declaration might equally be moved into the subroutine in which case the whole subproblem is recreated at every call to the function `knapsack`.

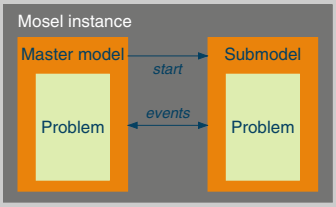
```
 declarations
  NWIDTHS = 5                          ! No of different widths
  WIDTHS = 1..NWIDTHS                  ! Range of widths

  Knapsack: mpproblem                  ! Knapsack subproblem
  KnapCtr, KnapObj: linctr             ! Knapsack constraint+obj.
  x: array(WIDTHS) of mpvar            ! Knapsack variables
 end-declarations

! ... Initialize the data

! Create integrality constraints for the knapsack subproblem
 with Knapsack do
  forall(j in WIDTHS) x(j) is_integer
 end-do
```

```
! ... State the main problem, start column generation algorithm

!**** Define and solve the knapsack subproblem ****
 function knapsack(C:array(range) of real,
                   A:array(range) of real,
                   B:real,
                   xbest:array(range) of integer):real

  with Knapsack do
 ! Redefine the knapsack problem
   KnapCtr := sum(j in WIDTHS) A(j)*x(j) <= B
   KnapObj := sum(j in WIDTHS) C(j)*x(j)

 ! Solve the problem and save the results
   maximize(KnapObj)
   returned:=getobjval
   forall(j in WIDTHS) xbest(j):=round(getsol(x(j)))
  end-do

 end-function
```

The solution values are returned in the argument `xbest` of the subroutine. The objective function value of the knapsack problem is the return value of the function `knapsack`.

The full description of this model can be found in the "Mosel User Guide" [6].

5.3.3 *Multi-Model*



Xpress-Mosel allows multiple optimization problems to be implemented as separate model (files). This approach is most suitable if the optimization process should be spread along several threads and executed in parallel. Xpress-Mosel's unique implementation characteristics make parallel and multithreaded optimization easily accessible. Readily available communication mechanisms include synchronization of concurrent models based on event queues and data exchange through shared memory.

The module *mmjobs* that forms part of the Mosel distribution enhances the Mosel language with functionality for loading several models in memory and executing

them concurrently. The synchronization mechanism is based on event queues. For the data exchange between concurrent models *mmjobs* provides shared memory and memory pipes I/O drivers.
Model management: the type `Model` represents a reference to a Mosel model that is initialized by loading a bim file (notice the spelling with upper case "M" to avoid a clash with the keyword `model`). The standard sequence for submodel execution is `compile-load-run-wait`. The submodel is started as a separate thread and we need to make sure that the submodel has terminated before the master model ends (or continues its execution with results from the submodel), `wait` pauses the execution of the master model until it receives an *event* (e.g. termination of the submodel).
Event handling: the type `Event` is characterized by a *class* (integer; reserved value: `EVENT_END`) and a *value* (real). Events are exchanged between a model and its immediate parent model. An event queue is attached to each model to collect all events sent to this model; this queue is managed with a FIFO policy (First In–First Out).
I/O drivers: the `shmem` shared memory I/O driver is a shared memory version of the `mem` driver for exchanging data between concurrent models (write access by a single model, read access by several models simultaneously). The `mempipe` memory pipe I/O driver implements memory pipes for exchanging data between concurrent models (write access by several models, read access by a single model). Both drivers can be used wherever Mosel expects a (generalized) filename, in particular in `initializations` blocks. The choice of the I/O driver provides an additional means of synchronizing concurrent models.

Here is an implementation of the knapsack subproblem from the previous section using a separate model file. The subroutine `knapsack` now triggers a run of the model `knapsack.mos` that has been compiled and loaded before the start of the column generation loop. Input and result data are communicated through the `shmem` driver, using binary format for highest accuracy.

```
 uses "mmjobs"

 declarations
  NWIDTHS = 5                      ! No. of different widths
  WIDTHS = 1..NWIDTHS              ! Range of widths

  Knapsack: Model                  ! Reference to knapsack model
 end-declarations

! ... Initialize the data

 res:= compile("knapsack.mos")     ! Compile the knapsack model
 load(Knapsack, "knapsack.bim")    ! Load the knapsack model

! ... State the main problem, start column generation algorithm

!**** Solve the knapsack subproblem ****
 function knapsack(C:array(range) of real,
```

```
                  A:array(range) of real,
                  B:real,
                  xbest:array(range) of integer):real

  INDATA  := "bin:shmem:indata"
  RESDATA := "bin:shmem:resdata"

  initializations to INDATA
   A  B  C
  end-initializations

  run(Knapsack, "NWIDTHS=" + NWIDTHS + ",INDATA=" + INDATA +
      ",RESDATA=" + RESDATA)      ! Start knapsack (sub)model
  wait                            ! Wait until model finishes
  dropnextevent                   ! Ignore termination message

  initializations from RESDATA
   xbest  returned as "zbest"
  end-initializations
 end-function
```

The model file `knapsack.mos` has the following content. By default, it uses physical data files, but the arguments of the `run` statement above overwrite these default filenames to work with data in shared memory.

```
model "Knapsack"
 uses "mmxprs"

 parameters
  NWIDTHS = 5                      ! No. of different widths
  INDATA = "knapsack.dat"          ! Input data file
  RESDATA = "knresult.dat"         ! Result data file
 end-parameters

 declarations
  WIDTHS = 1..NWIDTHS              ! Range of widths
  A,C: array(WIDTHS) of real       ! Coefficient arrays
  B: real                          ! RHS of knapsack constr.
  KnapCtr, KnapObj: linctr         ! Knapsack constraint+obj.
  x: array(WIDTHS) of mpvar        ! Knapsack variables
  xbest: array(WIDTHS) of integer  ! Solution values
 end-declarations

 initializations from INDATA
  A  B  C
 end-initializations

! Define the knapsack problem
 KnapCtr:= sum(j in WIDTHS) A(j)*x(j) <= B
 KnapObj:= sum(j in WIDTHS) C(j)*x(j)

! Integrality condition
 forall(j in WIDTHS) x(j) is_integer
```

```
! Solve the problem and save the results
 maximize(KnapObj)
 z:=getobjval
 forall(j in WIDTHS) xbest(j):=round(getsol(x(j)))

 initializations to RESDATA
  xbest  z as "zbest"
 end-initializations

end-model
```

Although slightly more complicated, this second implementation has the advantage over the version from the previous section that the knapsack subproblem can be run as a standalone model.

Using Mosel's parallel computing capacities, more complicated schemes of concurrent solving can be implemented. Figure 5.8 shows the simple scheme used for the knapsack subproblem. Figure 5.9 extends this scheme to concurrent submodel runs that are coordinated via *events* (= messages exchanged between parent and child models).

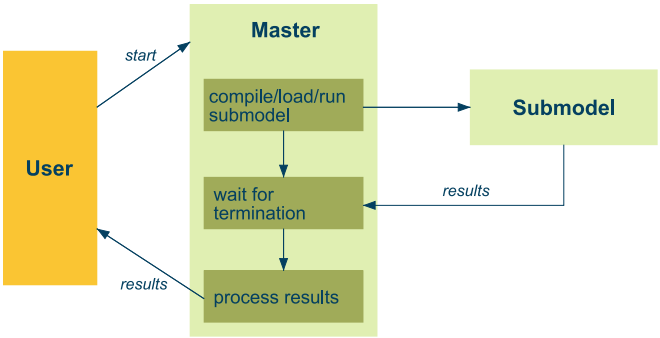


Fig. 5.8 Simple submodel call

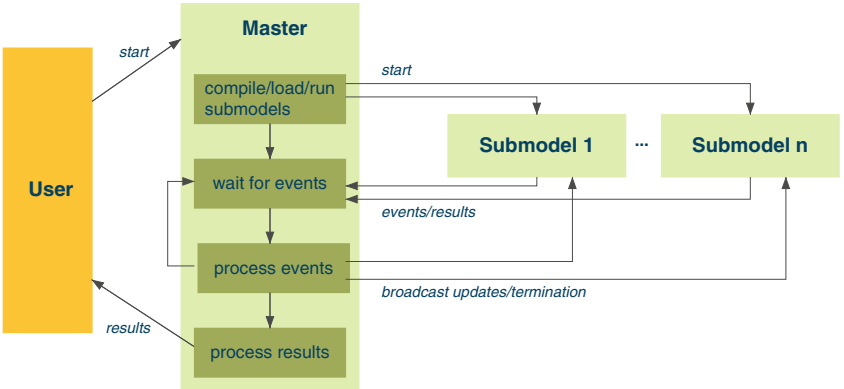


Fig. 5.9 Concurrent submodels coordinated via events

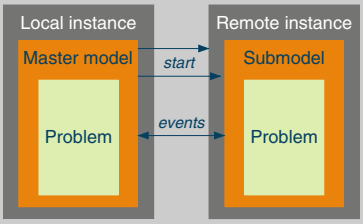
The functionality of *mmjobs* can be used to implement various schemes of decomposition and concurrent solving in Mosel, for instance:

- Simple parallel runs (different data instances; different algorithm configurations)
- Decomposition approaches (Benders; Dantzig-Wolfe)
- Column generation (loop over top node; branch-and-price)
- Cut generation (cut-and-branch, branch-and-cut)

Problem solving approaches that involve parallel execution of (sub)models can only be implemented as multiple models, whereas sequential solving can be formulated either way. For sequential algorithms the developer may choose between the two design options. For more detail on these techniques and complete implementation examples with Mosel we refer to the whitepaper "Multiple models and parallel solving with Mosel" [6].

When developing decomposition algorithms that involve concurrent solving of optimization subproblems on a multiprocessor computer we recommend that the total number of parallel threads be limited (taking into account possible parallel threads started by the solvers for each submodel) to the available number of processors.

5.3.4 Multi-Node



Mosel's capacities for handling multiple models have recently been extended to distributed computing using several Mosel instances (running locally or on remote nodes connected through a network). This facility opens new perspectives for the implementation of decomposition approaches, using all the computing power available in a local network. Moving from a multi-model application on a single instance to a multi-node application only requires a few changes in a model, not least thanks to the concept of I/O drivers—prefixes to the file name that indicate how to access a given file (remotely, in memory, compressed, etc.).

The model management and synchronzation mechanisms of module *mmjobs* have recently been extended to distributed computing using several *Mosel instances*

(running locally or on remote nodes connected through a network). The type `Mosel` represents a Mosel instance. Before loading and running a model on another instance, the new instance must be connected. The distributed computing functionality also comprises two remote connection IO drivers (`xsrv` and `rcmd`) for creating remote Mosel instances and the remote file acces IO driver `rmt` that is usable wherever Mosel expects a (generalized) filename, in particular in `initializations` blocks.

In order to be able to establish a connection, the remote machine must be running a server. The default server is the Mosel server *xprmsrv* that is started as a separate program, available for all platforms supported by Xpress, and connected to using driver `xsrv`:

```
connect(mosInst, "ABCD123")
```

Alternatively, other servers such as *ssh* for a secure connection can be connected to using driver `rcmd` (NB: the Mosel command line option `-r` is required for remote runs):

```
connect(mosInst, "rcmd:ssh ABCD123 mosel -r")
```

Instead of the machine name "ABCD123" we might have used the IP-address of the remote machine. The Mosel server can be configured. Configuration options include verbosity settings, choice of the TCP port, and the definition of a log file. The more flexible configuration through a *configuration file* makes it possible to define multiple *environments* that may include the choice of the Xpress version, definition of a working directory and security settings (read/write access rights, password protected access).

The multi-model version of the column generation algorithm from the previous section only requires a few changes in order to run the knapsack subproblem on a remote machine (Fig. 5.10): we need to establish a connection and load the compiled model into the remote Mosel instance. Data is held at the location of the local instance, the remote model therefore needs to use the `rmt` remote access driver for reading and writing data. There are no other changes required to the subroutine `knapsack` and the knapsack model itself remains entirely unchanged.

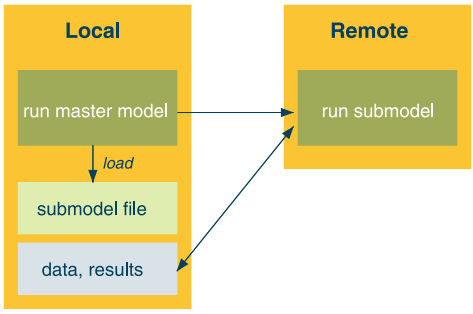


Fig. 5.10 Remote model execution with data on local machine

```
 uses "mmjobs"

 declarations
  NWIDTHS = 5                        ! No. of different widths
  WIDTHS = 1..NWIDTHS                ! Range of widths

  mosInst: Mosel                     ! Mosel instance
  Knapsack: Model                    ! Reference to knapsack model
 end-declarations

! ... Initialize the data

       !!! Use the name or IP address of a machine in
       !!! your local network, or "" for current node
 NODENAME:= ""
                         ! Compile the knapsack model
 if compile("knapsack.mos")<>0 then exit(1); end-if
                         ! Open connection to a remote node
 if connect(mosInst, NODENAME)<>0 then exit(2); end-if
                         ! Load submodel into the remote instance
 load(mosInst, Knapsack, "rmt:knapsack.bim")

! ... State the main problem, start column generation algorithm

!**** Solve the knapsack subproblem ****
 function knapsack(C:array(range) of real,
                   A:array(range) of real,
                   B:real,
                   xbest:array(range) of integer):real

  initializations to "bin:indata.dat"
   A  B  C
  end-initializations

  run(Knapsack, "NWIDTHS=" + NWIDTHS +
   ",INDATA=bin:rmt:indata.dat" +
   ",RESDATA=bin:rmt:resdata.dat")   ! Start knapsack (sub)model
  wait                               ! Wait until model finishes
  dropnextevent                      ! Ignore termination message

  initializations from "bin:resdata.dat"
   xbest  returned as "zbest"
  end-initializations

 end-function
```

The design of distributed Mosel applications can be adapted flexibly to the needs of a particular business environment. Most notably, there are no restrictions on the network architecture (any platform supported by Xpress can be used), a 32-bit Windows user frontend can be combined, for instance, with 64-bit Unix machines; it is even possible to work with several non-identical Mosel instances on a single

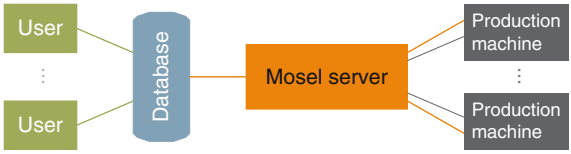


Fig. 5.11 Multi-user application with Mosel model as dispatcher

computer (using different software releases or combining 32 and 64-bit versions). Furthermore, the physical location of data and model files can be chosen freely (for instance, locally on the master node, on the nodes processing the models, or in a centralized repository).

Figure 5.11 outlines the design of a distributed Mosel application in a multi-user environment that is already in production use. Users configure *optimization model* instances that are stored as requests in a database. The center piece is a Mosel *dispatcher model* that checks the database and registers new requests into its model queues. Whenever a suitable production machine becomes available a new *optimization model* run is started. Results (or reasons for failure or interruption) are reported back to the users through the request database. Implementation effort for such a dedicated dispatcher model is quite moderate, involving less than 1,000 lines of Mosel code for implementing a dynamic model queuing system and all database interactions in this application.

For further details on distributed computing please refer to the whitepaper "Multiple models and parallel solving with Mosel" [6].

5.4 Summary

Ten years after its first commercialization, Xpress-Mosel has a proven track record of innovations, starting with its all-in-one design, the open interface for user modules, its profiler and debugger tools and most recently, the introduction of distributed computing features. The evolution of the Mosel language is always accompanied by new versions of the graphical development environment IVE. New solving functionality of the Xpress suite has been made available regularly through updates to the corresponding Mosel modules.

The Mosel environment has been used for several years in mission-critical and large-scale applications across all sectors of industry, not least thanks to its flexible integration and application design options. The recent addition of distributed computing facilities opens many further possibilities for its use.

The interested reader is refered to the examples and documentation on the Xpress webpage [5, 6] and the collection of Mosel model examples in the book "Applications of optimization with Xpress-MP" [8].

Appendix: Complete Mosel Examples

Solution Enumeration

The following Mosel model shows how to configure Xpress-Optimizer to use the solution enumerator for finding alternative MIP solutions, saving the N best solutions found and displaying all saved solutions after completion of the optimization run.

```
model "Portfolio optimization with MIP"

 uses "mmxprs"                  ! Use Xpress-Optimizer

 declarations
  SHARES: set of string         ! Set of shares
  RISK: set of string           ! High-risk values among shares
  NA: set of string             ! Shares issued in N.-America
  RET: array(SHARES) of real    ! Estim. return in investment
 end-declarations

 initializations from "folio.dat"
  RISK RET NA
 end-initializations

 declarations
  frac: array(SHARES) of mpvar  ! Fraction of capital per share
  buy: array(SHARES) of mpvar   ! 1 iff asset is in portfolio
 end-declarations

! Objective: total return
 Return:= sum(s in SHARES) RET(s)*frac(s)

! Limit the percentage of high-risk values
 sum(s in RISK) frac(s) <= 1/3

! Minimum amount of North-American values
 sum(s in NA) frac(s) >= 0.5

! Spend all the capital
 sum(s in SHARES) frac(s) = 1

! Upper bounds on the investment per share
 forall(s in SHARES) frac(s) <= 0.3

! Limit the total number of assets
 sum(s in SHARES) buy(s) <= 4

 forall(s in SHARES) do
  buy(s) is_binary              ! Turn variables into binaries
  frac(s) <= buy(s)             ! Linking the variables
 end-do
```

```
! Set the max. number of solutions to store (default: 10)
 setparam("XPRS_enummaxsol", 25)

! Solve the problem, enabling the solution enumerator
 maximize(XPRS_ENUM, Return)

! Print out all solutions saved by the enumerator
 forall(i in 1..getparam("XPRS_enumsols")) do
  selectsol(i)              ! Select a solution from the pool
  writeln("Solution ", i)
  print_sol
 end-do

! Solution printing
 procedure print_sol
  writeln("Total return: ", getobjval)
  forall(s in SHARES | getsol(frac(s))>0)
   writeln(s, ": ", getsol(frac(s))*100, "%")
 end-procedure
end-model
```

Column Generation Algorithm for the Cutting Stock Problem

The following is the complete code of the column generation algorithm for solving the cutting stock problem (master model of the implementation using a separate model file for the knapsack subproblem, see Section *Multi-Model*). For simplicity's sake, all input data is contained directly in the model.

```
model "Papermill (multi-prob)"
 uses "mmxprs", "mmjobs", "mmsystem"

 forward procedure column_gen
 forward function knapsack(C:array(range) of real,
                           A:array(range) of real,
                           B:real,
                           xb:array(range) of integer): real
 forward procedure show_new_pat(dj:real,
                                vx: array(range) of integer)
 declarations
  NWIDTHS = 5                     ! Number of different widths
  WIDTHS = 1..NWIDTHS             ! Range of widths
  RP: range                       ! Range of cutting patterns
  MAXWIDTH = 94                   ! Maximum roll width
  EPS = 1e-6                      ! Zero tolerance

  WIDTH: array(WIDTHS) of real    ! Possible widths
  DEMAND: array(WIDTHS) of integer  ! Demand per width
  PATTERNS: array(WIDTHS, WIDTHS) of integer
                                  ! (Basic) cutting patterns
  use: dynamic array(RP) of mpvar ! Rolls per pattern
```

```
  soluse: dynamic array(RP) of real ! Solution values for 'use'
  Dem: array(WIDTHS) of linctr    ! Demand constraints
  MinRolls: linctr                ! Objective function

  Knapsack: Model                 ! Reference to knapsack model
 end-declarations

 WIDTH::  [ 17, 21, 22.5, 24, 29.5]
 DEMAND:: [150, 96, 48, 108, 227]

                                  ! Make basic patterns
 forall(j in WIDTHS)  PATTERNS(j,j) := floor(MAXWIDTH/WIDTH(j))

 forall(j in WIDTHS) do
  create(use(j))                  ! Create NWIDTHS var.s 'use'
  use(j) is_integer               ! Variables are integer
  use(j) <= integer(ceil(DEMAND(j)/PATTERNS(j,j)))
 end-do

! Objective: Minimize total number of rolls
 MinRolls:= sum(j in WIDTHS) use(j)

! Constraints: Satisfy the demands
 forall(i in WIDTHS)
  Dem(i):= sum(j in WIDTHS) PATTERNS(i,j)*use(j) >= DEMAND(i)

 res:= compile("knapsack.mos")    ! Compile the knapsack model
 load(Knapsack, "knapsack.bim")   ! Load the knapsack model
 fdelete("knapsack.bim")          ! Cleaning up

 column_gen                       ! Start column generation

! Compute the best integer solution for the resulting current
! problem (including the new columns)
 minimize(MinRolls)

 writeln("Best integer solution: ", getobjval, " rolls")
 write("  Rolls per pattern: ")
 forall(i in RP) write(getsol(use(i)),", ")
 writeln

!*************************************************************
! Column generation loop at the top node:
!   * solve the LP and save the basis
!   * get the solution values
!   * generate new column(s) (=cutting pattern)
!   * load the modified problem and load the saved basis
!*************************************************************
 procedure column_gen
  declarations
   dualdem: array(WIDTHS) of real
   xbest: array(WIDTHS) of integer
   dw, zbest, objval: real
   bas: basis
```

```
  end-declarations

  defcut:= getparam("XPRS_CUTSTRATEGY") ! Save curr. setting
  setparam("XPRS_CUTSTRATEGY", 0)    ! Disable automatic cuts
  setparam("XPRS_PRESOLVE", 0)       ! Switch presolve off
  setparam("zerotol", EPS)           ! Comparison tolerance
  npatt:= NWIDTHS
  npass:= 1
  while(true) do
    minimize(XPRS_LIN, MinRolls)     ! Solve the LP

    savebasis(bas)                   ! Save the current basis
    objval:= getobjval               ! Get the objective value

                                     ! Get the solution values
    forall(j in 1..npatt)  soluse(j):=getsol(use(j))
    forall(i in WIDTHS)  dualdem(i):=getdual(Dem(i))
                                     ! Solve a knapsack problem
    zbest:= knapsack(dualdem, WIDTH, MAXWIDTH, DEMAND, xbest)-1

    write("Pass ", npass, ": ")
    if zbest = 0 then                ! Same as zbest <= EPS
      writeln("no profitable column found.\n")
      break
    else
      show_new_pat(zbest, xbest)     ! Print the new pattern
      npatt+=1
      create(use(npatt))             ! Create a new variable
      use(npatt) is_integer

      MinRolls+= use(npatt)          ! Add new var. to objective
      dw:=0
      forall(i in WIDTHS)
        if xbest(i) > 0 then
         Dem(i) += xbest(i)*use(npatt)  ! Add var. to constr.s
         dw:= maxlist(dw, ceil(DEMAND(i)/xbest(i) ))
        end-if
      use(npatt) <= dw               ! Set upper bound on new var.

      loadprob(MinRolls)             ! Reload the problem
      loadbasis(bas)                 ! Load the saved basis
    end-if
    npass+=1
  end-do

  writeln("Solution after column generation: ", objval,
          " rolls, ", getsize(RP), " patterns")
  write("   Rolls per pattern: ")
  forall(i in RP) write(soluse(i),", ")
  writeln

  setparam("XPRS_CUTSTRATEGY", defcut) ! Enable automatic cuts
  setparam("XPRS_PRESOLVE", 1)         ! Switch presolve on
 end-procedure
```

```
!*************************************************************
! Solve the integer knapsack problem
!    z = max{cx : ax<=b, x<=d, x in Z^N}
!    with b=MAXWIDTH, N=NWIDTHS, d=DEMAND
!
! Data is exchanged between the two models via shared memory.
!*************************************************************
 function knapsack(C:array(range) of real,
                   A:array(range) of real,
                   B:real,
                   xbest:array(range) of integer):real

  INDATA  := "bin:shmem:indata"
  RESDATA := "bin:shmem:resdata"

  initializations to INDATA
   A  B  C
  end-initializations

  run(Knapsack, "NWIDTHS=" + NWIDTHS + ",INDATA=" + INDATA +
      ",RESDATA=" + RESDATA)       ! Start knapsack (sub)model
  wait                             ! Wait until model finishes
  dropnextevent                    ! Ignore termination message

  initializations from RESDATA
   xbest  returned as "zbest"
  end-initializations

 end-function

!*************************************************************
! Display a generated pattern (= new column)
!*************************************************************
 procedure show_new_pat(dj:real, vx: array(range) of integer)
  declarations
   dw: real
  end-declarations

  writeln("New pattern found with marginal cost ", dj)
  write("   Widths distribution: ")
  dw:=0
  forall(i in WIDTHS) do
    write(WIDTH(i), ":", vx(i), "  ")
    dw += WIDTH(i)*vx(i)
  end-do
  writeln("Total width: ", dw)
 end-procedure

end-model
```

References

1. Ashford, R.W., Daniel, R.C.: LP-MODEL: XPRESS-LP's Model Builder. IMA J. Math. Manag. **1**, 163–176 (1987)
2. Chlond, M., Daniel, R.C., Heipcke, S.: Fiveleapers a-leaping. INFORMS Trans. Educ. **4**(1) (2003). URL http://ite.pubs.informs.org/Vol4No1/
3. Ciriani, T.A., Colombani, Y., Heipcke, S.: Embedding optimisation algorithms with Mosel. 4OR **1**(2), 155–168 (2003)
4. Colombani, Y., Daniel, B., Heipcke, S.: Mosel: A Modular Environment for Modeling and Solving Problems. In: J. Kallrath (ed.) Modeling Languages in Mathematical Optimization, pp. 211–238. Kluwer Academic Publishers, Boston (2004)
5. FICO Xpress Team: FICO Xpress Examples Repository (2011). URL http://examples.xpress.fico.com
6. FICO Xpress Team: Xpress online documentation (2011). URL http://optimization.fico.com/product-information
7. Fourer, R., Gay, D., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming. The Scientific Press, San Francisco, CA (1993)
8. Guéret, C., Heipcke, S., Prins, C., Sevaux, M.: Applications of Optimization with Xpress-MP. Dash Optimization, Blisworth, UK (2002). URL http://optimization.fico.com/product-information
9. Maximal Software: MPL User Manual (2011). URL http://www.maximal-usa.com
10. Van Hentenryck, P.: The OPL Optimization Programming Language. MIT Press, Cambridge, MA (1998)

Chapter 6
Interval-Based Language for Modeling Scheduling Problems: An Extension to Constraint Programming

Philippe Laborie, Jérôme Rogerie, Paul Shaw, Petr Vilím, and Ferenc Katai

Abstract IBM ILOG CP Optimizer (CPO) provides a scheduling language supported by a robust and efficient automatic search. This paper summes up the major language constructs and shed some lights on their propagations. Among the main constructs it introduces the notion of interval variable which enable reasoning on conditional time-intervals representing activities or tasks that may or may not be executed in the final schedule. In Constraint-Based Scheduling, those problems are usually handled by defining new global constraints over classical integer variables. This dual perspective facilitates an easy modeling process while ensuring a strong constraint propagation and an efficient search in the engine. The approach forms the foundations of the new generation of scheduling model and algorithms provided in CPO. Small examples are provided at the end of the language construct, however at the end of the paper three larger/real life examples recently studied in the scheduling literature are presented along some computational results illustrating both the expressivity of the modeling language and the robustness of the automatic search. Interestingly all three problems can easily be modeled with the language in only a few dozen lines (the complete models are provided) and on average the automatic search outperforms existing problem specific approaches.

6.1 Introduction

Many scheduling problems involve reasoning about activities or processes that may or may not be executed in the final schedule. This is particularly true as scheduling is evolving in the direction of AI Planning whose core problem is precisely the

P. Laborie (✉) · J. Rogerie · P. Shaw · P. Vilím · F. Katai
IBM, 9 rue de Verdun, F-94253 Gentilly Cedex, France
e-mail: laborie@fr.ibm.com; rogerie@fr.ibm.com; paul.shaw@fr.ibm.com; petr_vilim@cz.ibm.com; ferenc.katai@fr.ibm.com

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_6, © Springer-Verlag Berlin Heidelberg 2012

selection of the set of activities to be executed. Indeed, most industrial scheduling applications present at least some of the following features:

- Optional activities (operations, tasks) that can be left unperformed (with an impact on the cost) : typical examples are externalized, maintenance or control tasks.
- Activities that can be executed on a set of alternative resources (machines, manpower) with possibly different characteristics (speed, calendar) and compatibility constraints.
- Operations that can be processed in different temporal modes (for instance in series or in parallel).
- Alternative modes for executing a given activity, each mode specifying a particular combination of resources.
- Alternative processes for executing a given production order, a process being specified as a sequence of operations requiring resources.
- Hierarchical description of a project as a work-breakdown structure with tasks decomposed into sub-tasks, part of the project being optional (with an impact on the cost if unperformed), *etc*.

Modeling and solving these types of problems is an active topic in Constraint-Based Scheduling. Most of the current approaches are based on defining additional decision variables that represent the existence of an activity in the schedule [2, 3] or the index variable of the alternative resource/mode allocated to an activity [14, 16] and proposing new global constraints and associated propagation algorithms: *XorNode, PEX* in [3], DTP_{FD} in [14], *P/A Graphs* in [2], *alternative resource constraints* in [16].

CPO uses a different approach based on the idea that optional activities should be considered as first class citizen *variables* in the representation (we call them *interval variables*) and that the constraint propagation engine is extended to handle this new type of decision variable. Roughly speaking, it is the dual view compared with existing approaches: instead of defining new constraints over classical integer variables to handle optional activities, we introduce them as new variables in the engine. As we will see in the sequel of this paper, doing this offers several advantages:

- Modeling is easy because the notion of optionality is intrinsic to the concept of *interval variables*: there is no need for additional variables and complex meta-constraints.
- The model is very expressive and separates the temporal aspects from the logical ones.
- Constraint propagation is strong because the conditional domain maintained in *interval variables* naturally allows conjunctive reasoning between constraints.
- Most of the global constraints in Constraint-Based Scheduling can be extended to efficiently propagate on *interval variables*.

Sections 6.2–6.4 focus on the notion of *interval variable* and on the basic temporal and logical constraints between them. This concept forms the foundations of the new

generation of scheduling model and algorithms embedded in CPO [9]. Additional constraints available in CPO for modeling resources are presented in Sects. 6.5–6.7. The modeling language is a direct mapping of those concepts, it is illustrated on three scheduling problems in Sect. 6.9.

6.2 Conditional Interval Model

6.2.1 Usage and Rationale

Interval variables and constraints over them make it easy to capture the structure of complex scheduling problems (hierarchical description of the work-breakdown structure of a project, representation of optional activities, alternatives of modes, recipes or processes, etc.) in a well-defined CP paradigm.

The integer expressions are provided to constrain the different components of an interval variable (start, end, length). For instance the expression startOf(a, dv) returns the start of interval variable a when it is present and integer value dv if it is absent. Those expressions make it possible to mix interval variables with integer variables, global constraints and expressions.

6.2.2 Interval Variables

Note that in this article, if x denotes a decision variable of the problem, we denote with an underline $\underline{x}$ a fixed decision variable, that is, a decision variable whose domain is reduced to a singleton.

A **interval variable** a is a variable whose domain $dom(a)$ is a subset of $\{\perp\} \cup \{[s, e) | s, e \in \mathbb{Z}, s \leq e\}$.[1] An interval variable is said to be **fixed** if its domain is reduced to a singleton, i.e., if $\underline{a}$ denotes a fixed interval:

- Interval is **absent**: $\underline{a} = \perp$; or
- Interval is **present**: $\underline{a} = [s, e)$ with $s \leq e$. In this case, s and e are respectively the **start** and **end** of the interval and $l = e - s$ its **length**.

Absent interval variables have special meaning. Informally speaking, an absent interval variable is not considered by any constraint or expression on interval variables it is involved in. For example, if an absent interval variable a is used in a precedence constraint between interval variables a and b this constraint does not impact interval variable b. Each constraint specifies how it handles absent interval variables.

[1] Note that there is a small abuse of notation here as we allow $s = e$. This can be used to represent a zero length interval at value s even if in this case the interval $[s, e)$ itself is empty.

The semantics of constraints defined over interval variables is described by the properties that fixed intervals must have in order the constraint to be true. If a fixed interval $\underline{a}$ is present and such that $\underline{a} = [u, v)$, we will denote $s(\underline{a})$ its integer start date u, $e(\underline{a})$ its integer end date v and $d(\underline{a})$ its positive integer length $v - u$. The presence status $x(\underline{a})$ will be equal to 1. For a fixed interval that is absent, $x(\underline{a}) = 0$ and the start, end and length are meaningless.

By default interval variables are supposed to be present but they can be specified as being **optional** meaning that $\perp$ is part of the domain of the variable and thus, it is a decision of the problem to have the interval present or absent in the solution. Optional interval variables provide a powerful concept for efficiently reasoning with optional or alternative activities.

An example of declaration of a two dimensional array of interval variables in OPL can be found in Model 1, line 9. An example of declaration of an array of optional interval variables can be found in Model 2, line 7.

The following constraints on interval variables are introduced to model the basic structure of scheduling problems. Let a, a_i and b denote interval variables and z an integer variable:

- A **presence constraint** presenceOf(a) states that interval a is present, that is $a \neq \perp$. This constraint can be composed, for instance presenceOf(a) $\Rightarrow$ presenceOf(b) means that the presence of a implies the presence of b.
- A **precedence constraint** (*e.g.* endBeforeStart(a, b, z)) specifies a precedence between interval end-points with an integer or variable minimal distance z provided both intervals a and b are present.
- A **span constraint** span $(a, \{a_1, ..., a_n\})$ states that if a is present, it starts together with the first present interval in $\{a_1, ..., a_n\}$ and ends together with the last one. Interval a is absent if and only if all the a_i are absent.
- An **alternative constraint** alternative $(a, \{a_1, ..., a_n\})$ models an exclusive alternative between $\{a_1, ..., a_n\}$: if interval a is present then exactly one of intervals $\{a_1, ..., a_n\}$ is present and a starts and ends together with this chosen one. Interval a is absent if and only if all the a_i are absent.

6.2.3 Presence Constraints

Presence status of interval variables can be further restricted by logical constraints. The **presence constraint** presenceOf(a) states that a given interval variable must be present. The semantics of the presence constraint on a fixed interval $\underline{a}$ is:

$$\text{presenceOf}(\underline{a}) \Leftrightarrow \big(x(\underline{a}) = 1\big)$$

In the basic model described in this section, we only consider unary and binary logical constraints between presence statuses, that is, constraints of the form of 2-SAT clauses over presence statuses: $[\neg]$ presenceOf(a) or $[\neg]$ presenceOf(a) $\vee$

[¬] presenceOf(b). For example if a and b are two conditional intervals such that when interval a is present then b must be present too, it can be modeled by the constraint ¬ presenceOf(a) ∨ presenceOf(b).

An example of declaration of this type of logical constraints in OPL can be found in Model 3, line 32.

6.2.4 Precedence Constraints

The temporal constraint network consists of a Simple Temporal Network (STN) extended to the presence statuses. For instance, a precedence relation endBeforeStart(a, b) states that *if both intervals a and b are present* then the end of a must occur before the start of b.

The semantics of the relation $PC(\underline{a}, \underline{b}, z)$ on a pair of fixed intervals $\underline{a}, \underline{b}$ and for a delay value z depending on the precedence relation type PC is given on Table 6.1.

Note that in general, the delay z specified in a precedence constraint can be a variable of the problem rather than a fixed value. For simplicity, we assume in this paper that it is always a fixed value.

An example of declaration of precedence constraints in OPL can be found in Model 1, line 16.

6.2.5 Intensity Functions

Sometimes the intensity of "work" is not the same during the whole interval. For example let's consider a worker who does not work during weekends (his work intensity during weekends is 0%) and on Friday he works only for half a day (his intensity during Friday is 50%). For this worker, 7 man-days work will last for longer than just 7 days. In this example 7 man-days represent what we call the *size* of the interval: that is, the length of the interval would be if the intensity function was always at 100%. In CP Optimizer, this notion is captured by an **integer step function** that describes the instantaneous *intensity*—expressed as a percentage—of a work over time. An interval variable is associated with an **intensity function** and a **size**. The intensity function F specifies the instantaneous ratio between size

Table 6.1 Precedence constraints semantics

Relation	Semantics
startBeforeStart	$x(\underline{a}) \wedge x(\underline{b}) \Rightarrow s(\underline{a}) + z \leq s(\underline{b})$
startBeforeEnd	$x(\underline{a}) \wedge x(\underline{b}) \Rightarrow s(\underline{a}) + z \leq e(\underline{b})$
endBeforeStart	$x(\underline{a}) \wedge x(\underline{b}) \Rightarrow e(\underline{a}) + z \leq s(\underline{b})$
endBeforeEnd	$x(\underline{a}) \wedge x(\underline{b}) \Rightarrow e(\underline{a}) + z \leq e(\underline{b})$
startAtStart	$x(\underline{a}) \wedge x(\underline{b}) \Rightarrow s(\underline{a}) + z = s(\underline{b})$
startAtEnd	$x(\underline{a}) \wedge x(\underline{b}) \Rightarrow s(\underline{a}) + z = e(\underline{b})$
endAtStart	$x(\underline{a}) \wedge x(\underline{b}) \Rightarrow e(\underline{a}) + z = s(\underline{b})$
endAtEnd	$x(\underline{a}) \wedge x(\underline{b}) \Rightarrow e(\underline{a}) + z = e(\underline{b})$

and length. If an interval variable a is present, the intensity function enforces the following relation:

$$100 \times size(a) \leq \int_{start(a)}^{end(a)} F(t)\, dt < 100 \times (size(a) + 1)$$

By default, the intensity function of an interval variable is a flat function equal to 100%. In this case, the concepts of *size* and *length* are identical.

It may also be necessary to state that an interval cannot start, cannot end at or cannot overlap a set of fixed dates. CP Optimizer provides the following constraints for modeling it. Let a denote an interval variable and F an integer stepwise function.

- **Forbidden start constraint**. Constraint forbidStart(a, F) states that whenever interval a is present, it cannot start at a value t where $F(t) = 0$.
- **Forbidden end constraint**. Constraint forbidEnd(a, F) states that whenever interval a is present, it cannot end at a value t where $F(t-1) = 0$.
- **Forbidden extent constraint**. Constraint forbidExtent(a, F) states that whenever interval a is present, it cannot overlap a point t where $F(t) = 0$.

Integer expressions are provided to constrain the different components of an interval variable (start, end, length, size). For instance the expression startOf(a, dv) returns the start of interval variable a when a is present and returns integer value dv if a is absent (by default if argument dv is omitted it assumes $dv = 0$). Those expressions make it possible to mix interval variables with more traditional integer constraints and expressions.

6.2.6 Interval Composition Constraints

This section describes two constraints over a group of intervals and allow a hierarchical definition of the model by encapsulating a group of intervals into one high-level interval. Both constraints are hybrid in the sense that they combine logical and temporal aspects. Here is an informal definition of these constraints:

- **Span constraint**. The constraint span$(a_0, \{a_1, ..., a_n\})$ states that, if the interval a_0 is present, it spans over all present intervals from the set $\{a_1, .., a_n\}$. That is, interval a_0 starts together with the first present interval from $\{a_1, .., a_n\}$ and ends together with the last one. If interval a_0 is absent then none of intervals $\{a_1, .., a_n\}$ is present.
- **Alternative constraint**. The constraint alternative$(a_0, \{a_1, .., a_n\})$ models an exclusive alternative between $\{a_1, .., a_n\}$. If interval a_0 is present then exactly one of intervals $\{a_1, .., a_n\}$ is present and a_0 starts and ends together with this chosen one. If interval a_0 is absent then none of intervals $\{a_1, .., a_n\}$ is present.

More formally, let $\underline{a_0}, \underline{a_1}, ..., \underline{a_n}$ be a set of fixed interval variables.

The **span constraint** span($\underline{a_0}, \{\underline{a_1}, ..., \underline{a_n}\}$) holds if and only if:

$$\neg x(\underline{a_0}) \Leftrightarrow \forall i \in [1,n], \neg x(\underline{a_i})$$

$$x(\underline{a_0}) \Leftrightarrow \begin{cases} \exists i \in [1,n], x(\underline{a_i}) \\ s(\underline{a_0}) = \min_{i \in [1,n], x(\underline{a_i})} s(\underline{a_i}) \\ e(\underline{a_0}) = \max_{i \in [1,n], x(\underline{a_i})} e(\underline{a_i}) \end{cases}$$

An example of declaration of span constraint in OPL can be found in Model 3, line 37.

The **alternative intervals constraint** alternative($\underline{a_0}, \{\underline{a_1}, ..., \underline{a_n}\}$) holds if and only if:

$$\neg x(\underline{a_0}) \Leftrightarrow \forall i \in [1,n], \neg x(\underline{a_i})$$

$$x(\underline{a_0}) \Leftrightarrow \exists k \in [1,n] \begin{cases} x(\underline{a_k}) \\ s(\underline{a_0}) = s(\underline{a_k}) \\ e(\underline{a_0}) = e(\underline{a_k}) \\ \forall j \neq k, \neg x(\underline{a_j}) \end{cases}$$

An example of declaration of alternative constraint in OPL can be found in Model 2, line 12.

6.3 Complexity

Although both the logical constraint network (2-SAT) and the temporal constraint network (STN) are polynomially solvable frameworks, finding a solution to the basic model described above that combines the two frameworks is NP-Complete (even without alternative and span constraints). The proof is a direct consequence of the fact the model allows the expression of the temporal disjunction between two intervals a and b. Indeed, such a temporal disjunction can be modeled using 4 additional conditional intervals a_1, a_2, b_1 and b_2 with the following constraint set:

$$\begin{aligned}
\text{startAtStart}(a, a_1); \quad & \text{startAtStart}(a, a_2); \\
\text{startAtStart}(b, b_1); \quad & \text{startAtStart}(b, b_2); \\
\text{endAtEnd}(a, a_1); \quad & \text{endAtEnd}(a, a_2); \\
\text{endAtEnd}(b, b_1); \quad & \text{endAtEnd}(b, b_2); \\
\text{endBeforeStart}(a_1, b_1); \quad & \text{endBeforeStart}(b_2, a_2); \\
\neg\,\text{presenceOf}(a_1) \vee \neg\,\text{presenceOf}(a_2); \quad & \text{presenceOf}(a_1) \vee \text{presenceOf}(a_2); \\
\neg\,\text{presenceOf}(b_1) \vee \neg\,\text{presenceOf}(b_2); \quad & \text{presenceOf}(b_1) \vee \text{presenceOf}(b_2); \\
\neg\,\text{presenceOf}(a_1) \vee \text{presenceOf}(b_1); \quad & \text{presenceOf}(a_1) \vee \neg\,\text{presenceOf}(b_1);
\end{aligned}$$

6.4 Graphical Conventions and Basic Examples

The following sections illustrate some examples of models. We are using the following graphical conventions:

- Interval variables are represented by a box. When necessary, the interval length is specified inside the box. A dotted box represents an optional interval variable.
- Temporal constraints are represented by plain arrows. Depending on the type of temporal constraint, the end-points of the arrow are connected with the appropriate time-points of the interval variables.
- Logical constraint are represented by dotted arrows and denote an implication relation (for instance presenceOf(a) $\Rightarrow$ presenceOf(b)). In case the arrow starts or ends at a cross, it means the corresponding end-point of the implication is the negation of the presence status (e.g. presenceOf(a) $\Rightarrow \neg$ presenceOf(b)).
- Span constraints are represented by a box (the spanning interval variable) containing the set of spanned interval variables.
- Alternative constraints are represented by a multi-edge labelled by XOR.

6.4.1 Alternative Modes with Compatibility Constraints

Suppose an activity a can be executed in n possible modes $\{a_i\}_{i\in[1,n]}$ with duration da_i for mode a_i and an activity b can be executed in m possible modes $\{b_j\}_{j\in[1,m]}$ with duration db_j for mode b_j. Furthermore, there are some mode incompatibility constraints (i, j) specifying that mode a_i for a is incompatible with mode b_j for b. This model is represented on Fig. 6.1, incompatibilities (i, j) are modeled by implications presenceOf(a_i) $\Rightarrow \neg$ presenceOf(b_j) ($\neg$ presenceOf(a_i)$\vee$ $\neg$ presenceOf(b_j)). Of course, in practical applications, interval variables a_i and b_j will require some conjunction of resources but this is out of the scope of the present section. If binary logical constraints are insufficient to model the compatibility rules, the presence status of the interval presenceOf(a) can also be used in standard constraint programming constructs such as n-ary logical or arithmetic expressions or table constraints [4].

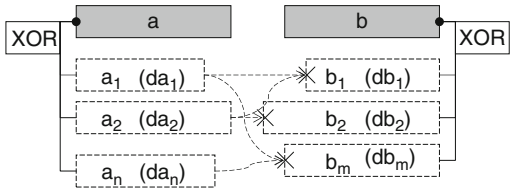


Fig. 6.1 A model for alternative modes

6.4.2 Series/Parallel Alternative

Figure 6.2 depicts a model where a job is composed of two operations a and b that can be executed either in series or in parallel (in this case, both operations are constrained to start at the same date). This is modeled by two alternatives alternative$(a, \{a_1, a_2\})$ and alternative$(b, \{b_1, b_2\})$ with (a_1, b_1) describing the serial and (a_2, b_2) describing the parallel execution. Logical constraints presenceOf$(a_i) \Leftrightarrow$ presenceOf(b_i) are added to ensure a consistent selection of the alternative.

A similar pattern can be used for any disjunction of a combination of temporal constraints on a pair of time interval variables.

6.4.3 Alternative Recipes

Figure 6.3 describes a set of 3 alternative recipes. The global process a is modeled as an alternative of the 3 recipes r_1, r_2 and r_3. Each recipe is a spanning interval variable that spans the internal operations of the recipe. Implication constraints between a recipe and some of its internal operations (for instance presenceOf$(r_3) \Rightarrow$ presenceOf(o_{31})) mean that operation is not optional in the recipe. Note that the opposite implications (for instance presenceOf$(o_{31}) \Rightarrow$ presenceOf(r_3)) are part of the span constraint. This pattern can be extended to a hierarchy of spanning tasks which is very convenient for modeling complex work-breakdown structures in project scheduling.

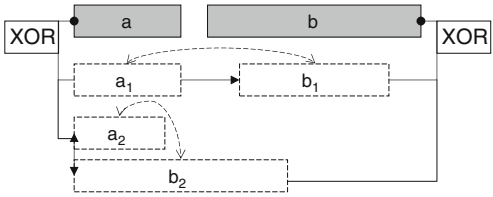


Fig. 6.2 A model for a series/parallel alternative

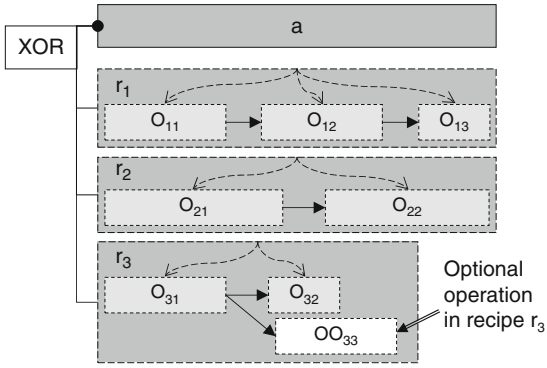


Fig. 6.3 A model for alternative recipes

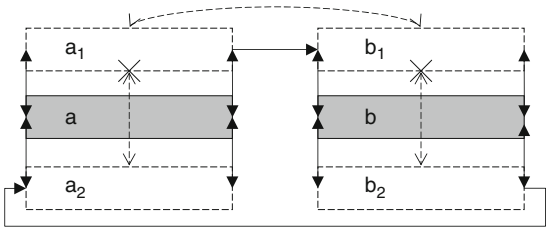


Fig. 6.4 A model for temporal disjunction

6.4.4 Temporal Disjunction

The graphical representation of the model for temporal disjunction which shows that the basic framework is NP-Complete as depicted in Fig. 6.4.

6.5 Sequence Variables

6.5.1 Usage and Rationale

Many scheduling problems involve disjunctive resources which can only perform one activity at a time (typical examples are workers, machines or vehicles). From the point of view of the resource, a solution is a sequence of activities to be processed. Besides the fact that activities in the sequence do not overlap in time, common additional constraints on such resources are setup times or constraints on the relative position of activities in the sequence.

To capture this idea we introduce the notion of *sequence variable*, a new type of decision variable whose value is a permutation of a set of interval variables. Constraints on interval variables are provided for ruling out illegal permutations (sequencing constraints) or for stating a particular relation between the order of intervals in the permutation and the relative position of their start and end values (no-overlap constraint).

6.5.2 Formal Semantics

6.5.2.1 Sequence Variable

A **sequence variable** p is defined on a set of interval variables A. Informally speaking, a value of p is a permutation of all present intervals of A. Let $n = |A|$ and $\underline{A}$ be an instantiation of the intervals of A. A permutation π of $\underline{A}$ is a function $\pi : \underline{A} \rightarrow [0, n]$ such that, if we denote $\text{length}(\pi) = |\{\underline{a} \in \underline{A}, x(\underline{a})\}|$ the number of present intervals:

1. $\forall \underline{a} \in \underline{A}, (\underline{a} = \perp) \Leftrightarrow (\pi(\underline{a}) = 0)$
2. $\forall \underline{a} \in \underline{A}, \pi(\underline{a}) \leq \text{length}(\pi)$
3. $\forall \underline{a}, \underline{b} \in \underline{A}, \pi(\underline{a}) = \pi(\underline{b}) \Rightarrow (\underline{a} = \underline{b} = \perp) \vee (a = b)$

For instance, if $A = \{a, b\}$ is a set of two interval variables with a being present and b optional, the domain of the sequence p defined on A consists of 3 values: $\{(a \rightarrow 1, b \rightarrow 0), (a \rightarrow 1, b \rightarrow 2), (a \rightarrow 2, b \rightarrow 1)\}$ or in short $\{(a), (a, b), (b, a)\}$.

6.5.2.2 Sequencing Constraints

The sequencing constraints below are available:

- First(p, a) states that if interval a is present then, it will be the first interval of the sequence p: $(\underline{a} \neq \perp) \Rightarrow (\pi\,(\underline{a}) = 1)$.
- Last(p, a) states that if interval a is present then, it will be the last interval of the sequence p: $(\underline{a} \neq \perp) \Rightarrow (\pi\,(\underline{a}) = \text{length}\,(\pi))$.
- Before(p, a, b) states that if both intervals a and b are present then a will appear before b in the sequence p: $(\underline{a} \neq \perp) \wedge (\underline{b} \neq \perp) \Rightarrow (\pi\,(\underline{a}) < \pi\,(\underline{b}))$.
- Prev(p, a, b) states that if both intervals a and b are present then a will be just before b in the sequence p, that is, it will appear before b and no other interval will be sequenced between a and b in the sequence p: $(\underline{a} \neq \perp) \wedge (\underline{b} \neq \perp) \Rightarrow (\pi\,(\underline{a}) + 1 = \pi\,(\underline{b}))$.

In the previous example, a constraint prev(p, a, b) would rule out value (b, a) as an illegal value of sequence variable p.

6.5.2.3 Transition Distance

Let $m \in \mathbb{Z}^+$, a **transition distance** is a function $M : [1, m] \times [1, m] \rightarrow \mathbb{Z}^+$. Transition distances are typically used to express a minimal delay that must elapse between two successive non-overlapping intervals.

6.5.2.4 No-Overlap Constraint

Note that the sequencing constraints presented above do not have any impact on the start and end values of intervals, they only constrain the possible values of the sequence variable. The **no-overlap constraint** on an interval sequence variable p states that the sequence defines a chain of non-overlapping intervals, any interval in the chain being constrained to end before the start of the next interval in the chain. A set of non-negative integer types $T(p, a)$ can be associated to each interval of a sequence variable. If a transition distance M is specified, it defines the minimal non-negative distance that must separate every two intervals in the sequence. More

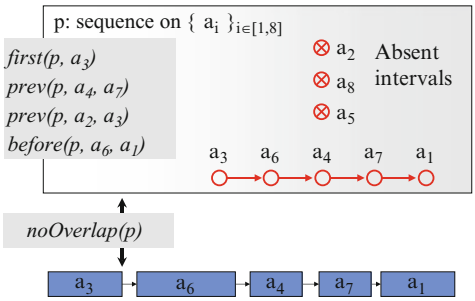


Fig. 6.5 Example of sequence variables and constraints

formally, let p be a sequence and let $T(p, a)$ be the type of interval a in sequence variable p, the condition for a permutation value π to satisfy the *no-overlap* constraint on p with transition distance M is defined as:

$$\text{noOverlap}(\pi, M) \Leftrightarrow \forall \underline{a}, \underline{b} \in \underline{A},$$
$$0 < \pi(\underline{a}) < \pi(\underline{b}) \Leftrightarrow e(\underline{a}) + M[T(p, \underline{a}), T(p, \underline{b})] \leq s(\underline{b})$$

Figure 6.5 illustrates the value of a sequence variable with a set of constraints it satisfies.

A simple example of declaration of no-overlap constraint in OPL can be found in Model 1, line 19. A slightly more complex example with transition distance is available in Model 3, line 41.

6.6 Cumul Function Expressions

6.6.1 *Usage and Rationale*

In scheduling problems involving cumulative resources, the cumulated usage of the resource by the activities is usually represented by a function of time. An activity increases the cumulated resource usage function at its start time and decreases it when it releases the resource at its end time. For resources that can be produced and consumed by activities (for instance the content of an inventory or a tank), the resource level can also be described as a function of time: production activities will increase the resource level whereas consuming activities will decrease it. In these problem classes, constraints are imposed on the evolution of these functions of time, for instance a maximal capacity or a minimum safety level. CP Optimizer introduces the notion of a *cumul function expression* which is a constrained expression that represents the sum of individual contributions of intervals.[2] A set of elementary

[2]In the rest of the paper, we often drop "expression" from "cumul function expression" to increase readability.

cumul functions is available to describe the individual contribution of an interval variable or a fixed interval of time. These elementary functions cover the use-cases mentioned above: *pulse* for usage of a cumulative resource, and *step* for resource production/consumption. When the elementary cumul functions that define a cumul function are fixed (and thus, so are their related intervals), the cumul function itself is fixed and its value is a stepwise integer function. Several constraints are provided over cumul functions. These constraints allow restricting the possible values of the function over the complete horizon or over some fixed or variable interval.

6.6.2 Formal Semantics

Let $\mathscr{F}^+$ denote the set of all functions from $\mathbb{Z}$ to $\mathbb{Z}^+$. A *cumul function expression* f is an expression whose value is a function of $\mathscr{F}^+$. Let $u, v \in \mathbb{Z}$ and $h, h_{min}, h_{max} \in \mathbb{Z}^+$ and a be an interval variable, we consider **elementary cumul functions** illustrated in Fig. 6.6.

Whenever the interval variable of an elementary cumul function is absent, the function is the zero function. A **cumul function** f is an expression built as the algebraic sum of the elementary functions of Fig. 6.6 or their negations. More formally, it is a construct of the form $f = \sum_i \epsilon_i \cdot f_i$ where $\epsilon_i \in \{-1, +1\}$ and f_i is an elementary cumul function.

The following constraints can be expressed on a cumul function f to restrict its possible values:

- alwaysIn($f, u, v, h_{min}, h_{max}$) means that the values of function f must remain in the range $[h_{min}, h_{max}]$ everywhere on the interval $[u, v)$.

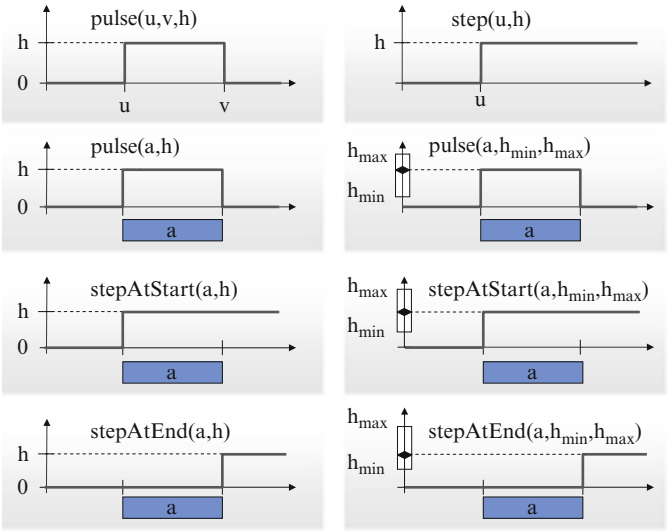


Fig. 6.6 Elementary cumul function expressions

- alwaysIn(f, a, h_{min}, h_{max}) means that if interval a is present, the values of function f must remain in the range $[h_{min}, h_{max}]$ between the start and the end of interval variable a.
- $f \leq h$: function f cannot take values greater than h.
- $f \geq h$: function f cannot take values lower than h.

An integer expression is introduced to get the total contribution of an interval variable a to a cumul function f at its start: heightAtStart(a, f, dh) with a default value dh in case a is absent. A similar expression exists for the end point. These expressions are useful to constrain the variable height of an elementary cumul function specified as a range $[h_{min}, h_{max}]$ using classical constraints on integer expressions.

6.6.3 Example

The constraints below model (1) a set of n activities $\{a_i\}$ such that no more than 3 activities in the set can overlap and (2) a chain of optional interval variables w_j that represent the distinct time-windows during which at least one activity a_i must execute. The constraints on interval variable status ensure that only the first intervals in the chain are present and the two alwaysIn constraints state the synchronization relation between intervals a_i and intervals w_j. A solution is illustrated on Fig. 6.7.

$$f_a = \sum_{i=1}^{n} \text{pulse}(a_i, 1); \quad f_w = \sum_{j=1}^{n} \text{pulse}(w_j, 1);$$

$$f_a \leq 3;$$

$$\forall j \in [1, n-1] \quad \begin{cases} \text{presenceOf}(w_{j+1}) \Rightarrow \text{presenceOf}(w_j); \\ \text{endBeforeStart}(w_j, w_{j+1}); \end{cases}$$

$$\forall i \in [1, n] \quad \begin{cases} \text{alwaysIn}(f_a, w_i, 1, n); \\ \text{alwaysIn}(f_w, a_i, 1, 1); \end{cases}$$

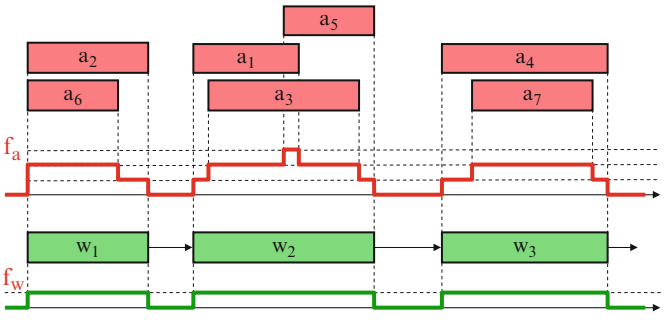


Fig. 6.7 Covering chain

An example of constrained cumul function expression in OPL can be found in Model 2, line 14.

6.7 State Function Variables

6.7.1 Usage and Rationale

In the same way as the value of an integer variable may represent an ordinal integer, functions over ordinal integers are useful in scheduling to describe the time evolution of a state variable. Typical examples are the time evolution of an oven's temperature, of the type of raw material present in a tank or of the tool installed on a machine. To that end, we introduce the notion of a *state function variable* and a set of constraints similar to the alwaysIn constraints on cumul functions to constrain the values of the state function.

A state function is a set of non-overlapping segments[3] over which the function maintains a constant non-negative integer state. In between those segments, the state of the function is not defined. For instance for an oven with 3 possible temperature levels identified by indices 0, 1 and 2 we could have the following time evolution (see also Fig. 6.8):

$$
\begin{aligned}
&[\text{start} = 0, \quad \text{end} = 100): \quad \text{state} = 0,\\
&[\text{start} = 140, \quad \text{end} = 300): \quad \text{state} = 1,\\
&[\text{start} = 320, \quad \text{end} = 500): \quad \text{state} = 2,\\
&[\text{start} = 540, \quad \text{end} = 600): \quad \text{state} = 2, \cdots
\end{aligned}
$$

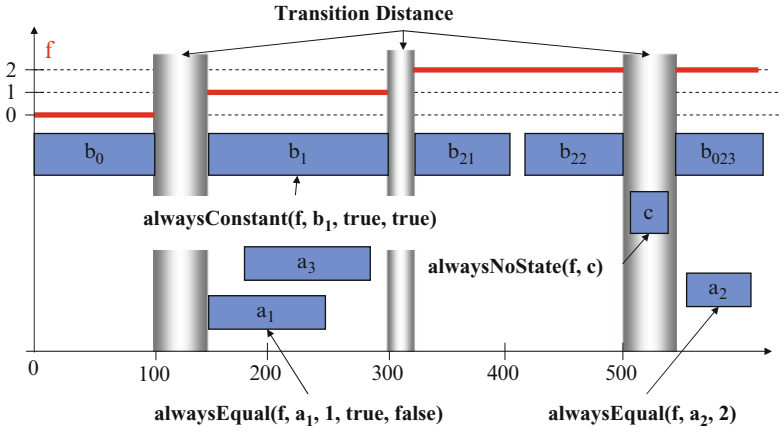


Fig. 6.8 State function

[3] A segment is an interval of integer. In the description of state functions we do not use the term *interval* to avoid confusion with interval variables.

6.7.2 Formal Semantics

6.7.2.1 State Function Variable

A **state function variable** f is a variable whose value is a set of non-overlapping segments, each segment $[s_i, e_i)$ (with $s_i < e_i$) is associated with a non-negative integer value v_i that represents the state of the function over the segment. Let $\underline{f}$ be a fixed state function, we will denote $\underline{f} = (\ [s_i, e_i)\ :\ v_i\ )_{i \in [1,n]}$. We denote $D(\underline{f}) = \cup_{i \in [1,n]}\ [s_i, e_i)$ the definition domain of $\underline{f}$, that is, the set of points where the state function is associated a state. For a fixed state function $\underline{f}$ and a point $t \in D(\underline{f})$, we will denote $[s(\underline{f}, t), e(\underline{f}, t))$ the unique segment of the function that contains t and $\underline{f}(t)$ the value of this segment. For instance, in the oven example we would have $\underline{f}(200) = 1$, $s(\underline{f}, 200) = 140$ and $e(f, 200) = 300$.

A state function can be endowed with a **transition distance**. The transition distance defines the minimal distance that must separate two consecutive states in the state function. More formally, if $M[v, v']$ is a transition distance matrix between state v and state v', we have: $\forall i \in [1, n-1],\ e_i + M[v_i, v_{i+1}] \leq s_{i+1}$.

6.7.2.2 Constraints on State Functions

If f is a state function of definition domain $D(f)$, a an interval variable, v, $v_{min} \leq v_{max}$ non-negative integers and $algn_s$, $algn_e$ two boolean values:

- The constraint alwaysConstant$(f, a, algn_s, algn_e)$ specifies that whenever a is present, the function takes a constant value between the start and the end of a. Boolean parameters $algn$ allow specifying whether or not interval variable a is synchronized with the start (resp. end) of the state function segment:

  (a) $[s(\underline{a}), e(\underline{a})) \subset [s(\underline{f}, s(\underline{a})), e(\underline{f}, s(\underline{a})))$
  (b) $algn_s \Rightarrow s(\underline{a}) = s(\underline{f}, s(\underline{a}))$
  (c) $algn_e \Rightarrow e(\underline{a}) = e(\underline{f}, e(\underline{a}))$
  (d) $\exists v \in \mathbb{Z}^+, \forall t \in [s(\underline{a}), e(\underline{a})),\ \underline{f}(t) = v$

- The constraint alwaysEqual$(f, a, v, algn_s, algn_e)$ specifies that whenever a is present the state function takes a constant value v over interval a:

  (a) alwaysConstant$(\underline{f}, \underline{a}, algn_s, algn_e)$
  (b) $v = \underline{f}(s(\underline{a}))$

- The constraint alwaysNoState(f, a) specifies that if a is present, it must not intersect the definition domain of the function, $[s(\underline{a}), e(\underline{a})) \cap D(\underline{f}) = \emptyset$.
- The constraint alwaysIn(f, a, v_{min}, v_{max}) where $0 \leq v_{min} \leq v_{max}$ specifies that whenever a is present, $\forall t \in [s(\underline{a}), e(\underline{a})) \cap D(\underline{f}),\ \underline{f}(t) \in [v_{min}, v_{max}]$.

Those constraints are also available on a fixed interval [start, end) as well as on an interval variable.

On Fig. 6.8, interval variables $b_0 : [0, 100)$ and $b_1 : [140, 300)$ are start and end aligned and thus, define two segments of the state function (of respective state 0 and 1). A transition distance 40 applies in between those states. Interval variable b_{21} is start aligned and interval b_{22} is end aligned both of state 2. As the transition distance $2 \rightarrow 2$ is greater than $s(b_{22}) - e(b_{21})$, the state function is aligned on $[s(b_{21}), e(b_{22}) = [320, 500)$. Interval variable c is constrained to be scheduled in a segment where the function is not defined. Finally, interval variables a_1, a_2 and a_3 require a particular state of the function, possibly with some alignment constraint as for a_1.

6.7.3 Example

The problem is to cook n items with an oven, each item $i \in [1, n]$ being cooked at a specific temperature v_i and for a specific range of duration. Items that are compatible both in temperature and in duration can be batched together and cooked simultaneously. Between two batches, a delay must elapse for cooling, emptying, loading, and heating the oven. For energy saving reasons the maximum reachable temperature is limited by v_{sup} over some time periods. The oven can be modeled as state function with a transition distance M. Each item is an interval variable a_i, possibly optional if the problem is over-constrained so that not all items can be cooked, and states an alwaysEqual constraint with start and end alignment. Each energy saving window is a fixed interval $[s_j, e_j)_{j \in [1,m]}$ that states an alwaysIn constraint:

$$\forall i \in [1, n], \text{alwaysEqual}(oven, a_i, v_i, true, true);$$
$$\forall j \in [1, m], \text{alwaysIn}(oven, s_j, e_j, 0, v_{sup});$$

6.8 Constraint Propagation and Search

6.8.1 Interval Variables

The domain of a interval variable a is represented by a tuple of ranges:

$$([x_{min}, x_{max}], [s_{min}, s_{max}], [e_{min}, e_{max}], [d_{min}, d_{max}])$$

$[x_{min}, x_{max}] \subseteq [0, 1]$ represents the domain of the presence status of a. $x_{min} = x_{max} = 1$ means that a will be present whereas $x_{min} = x_{max} = 0$ means that it will not be. $[s_{min}, s_{max}] \subseteq \mathbb{Z}$ represents the conditional domain of the start time of a, that is, the minimal and maximal start time *would a be present*. Similarly, $[d_{min}, d_{max}] \subseteq \mathbb{Z}$ and $[e_{min}, e_{max}] \subseteq \mathbb{Z}$ respectively denote the conditional domain of the length and end time of a.

The interval variable maintains the internal consistency between the temporal bounds $[s_{min}, s_{max}]$, $[e_{min}, e_{max}]$ and $[d_{min}, d_{max}]$ that are due to the relation $d = e - s$. When the temporal bounds become inconsistent (for instance because $s_{min} > s_{max}$ or because $e_{min} - s_{max} > d_{max}$), the interval presence status is automatically set to false ($x_{max} = 0$). Of course, if presence status of the interval is already true ($x_{min} = 1$), this will trigger a failure. Just like other classical variables in CSPs:

- The domain of interval variables can be accessed thanks to accessors: *isPresent*, *isAbsent*, *get*[*Start*|*End*|*Length*][*Min*|*Max*].
- It can be modified thanks to *modifiers*: *setPresent*, *setAbsent*, *set*[*Start*|*End*|*Length*][*Min*|*Max*].
- Events can be attached to the change of the domain so as to trigger constraint propagation. In this context, accessors are also available to access the previous value of the domain which is useful for implementing efficient incremental constraints: *getOld*[*Start*|*End*|*Length*][*Min*|*Max*].

The following sections describe how logical, temporal and hybrid n-ary constraints are handled by the engine.

6.8.2 Logical Network

All 2-SAT logical constraints between interval presence statuses of the form $[\neg]$ presenceOf$(a) \vee [\neg]$ presenceOf(b) are aggregated in a *logical network* similar to the implication graph described in [5]. The objectives of the logical network are:

- The detection of inconsistencies in logical constraints.
- An $O(1)$ access to the logical relation that can be inferred between any two intervals (a, b).
- A traversal of the set of intervals whose presence is implied by (resp. implies) the presence of a given interval variable a.
- The triggering of some events as soon as a new implication relation is inferred between two intervals (a, b) in order to wake up constraint propagation.

The above services are incrementally ensured when new logical constraints are added to the network or when the presence status of a interval is fixed. They are used by the propagation—as for instance in the temporal network presented in next section—and the search algorithms.

Nodes $\{l_i\}_{i \in [1,n]}$ in the graph correspond to presence statuses presenceOf(a) or their negation $\neg$ presenceOf(a) and an arc $l_i \rightarrow l_j$ corresponds to an implication relation between the corresponding boolean statuses. For instance a constraint presenceOf$(a) \vee$ presenceOf(b) would be associated with an arc $\neg$ presenceOf$(a) \rightarrow$ presenceOf(b). As links $l_i \rightarrow l_j$ and $\neg l_j \rightarrow \neg l_i$ are equivalent, for each interval variable a, only one node has to be considered in the network, either the one corresponding to presenceOf(a) or the one corresponding to $\neg$ presenceOf(a). Fixed presence statuses are skipped from the network as in this case binary constraints are reduced to unary constraints. Strongly connected components of

the implication graph are collapsed into a single node representing the logical equivalence class and the transitive closure of the resulting directed acyclic graph is maintained as new arcs are added. The logical network becomes inconsistent when it allows to infer both relations $l_i \rightarrow \neg l_i$ and $\neg l_i \rightarrow l_i$.

The time and memory complexity of the logical network for performing the transitive closure is quadratic with the length of the implication graph. In usual scheduling problems, this length tends to be small compared with the number of interval variables. Typically, this length is related with the depth of the work-breakdown structure.

6.8.3 Temporal Network

All temporal constraints of the form [*starts*|*ends*][*Before*|*At*][*Start*|*End*] are aggregated in a *temporal network* whose nodes $\{t_i\}_{i \in [1,n]}$ represent the set of interval start and end time-points. If t_i is a time-point in the network, we denote $x(t_i)$ the (variable) boolean presence status of the interval variable of t_i and $d(t_i)$ the (variable) date of the time-point. An arc (t_i, t_j, z_{ij}) in the network denotes a minimal delay z_{ij} between the two time-points t_i and t_j *would both time-points be present*, that is: $x(t_i) \wedge x(t_j) \Rightarrow (d(t_i) + z_{ij} \leq d(t_j))$. It is easy to see that all temporal constraints can be represented by one or two arcs in the temporal network. Furthermore, the length of the interval is also represented by two arcs, one between the start and the end time-point labelled with the minimal length and the other between the end and the start time-point labelled by the opposite of the maximal length. Let $d_{min}(t_i)$ and $d_{max}(t_i)$ denote the current conditional bounds on the date of time-point t_i. Depending on whether t_i denotes the start or the end of a interval variable, these bounds are the s_{min}, s_{max} or the e_{min}, e_{max} values stored in the current domain of the interval variable of t_i.

The main idea of the propagation of the temporal network is that for a given arc (t_i, t_j, z_{ij}), whenever the logical network can infer the relation $x(t_i) \Rightarrow x(t_j)$ the propagation on the conditional bounds of t_i (time-bounds *would* t_i *be present*) can assume that t_j will also be present and thus the arc can propagate the conditional bounds from time-point t_j on t_i: $d_{max}(t_i) \leftarrow \min(d_{max}(t_i), d_{max}(t_j) - z_{ij})$. Similarly, if the relation $x(t_j) \Rightarrow x(t_i)$ can be inferred by the logical network then the other half of the propagation that propagates on time-point t_j can be performed: $d_{min}(t_j) \leftarrow \max(d_{min}(t_j), d_{min}(t_i) + z_{ij})$. This observation is crucial: it allows to propagate on the conditional bounds of time-points even when their presence status is not fixed. Of course, when the presence status of a time-point t_i is fixed to 1, all other time-points t_j are such that $x(t_j) \Rightarrow x(t_i)$ and thus, the bounds of t_i can be propagated on all the other time-points. When all time-points are surely present, this propagation boils down to the classical bound-consistency on STNs. When the two time-points of an arc have equivalent presence status, the arc can be propagated in both directions, this is in particular the case for arcs corresponding to interval lengths. When the time-bounds of the extremities of an arc (t_i, t_j, z_{ij})

become inconsistent, the logical constraint $x(t_i) \Rightarrow \neg x(t_j)$ can be added to the logical network.

Figure 6.9 depicts the problem described in [2] modeled in our framework. If the deadline for finishing the schedule is 70, the propagation will infer that the alternative $BuyTube$ cannot be present as there is not enough space between the minimal start time of $GetTube$ (1) and its maximal end time (50) to accommodate its duration of 50. Note that if the duration of operation $BuyTube$ was lower than 49 but if the sum of the durations of operations $SawTube$ and $ClearTube$ was greater than 49, then the propagation would infer that the alternative $SawTube \rightarrow ClearTube$ is impossible because these two operations have equivalent presence status and thus, the precedence arc between them can propagate in both directions.

Figure 6.10 illustrates another example of propagation. The model consists of a chain of n identical optional operations $\{o_i\}_{i \in [1,n]}$ of duration 10 that, if present, need to be executed before date 25 and are such that presenceOf$(o_{i+1}) \Rightarrow$ presenceOf(o_i). Although initially, all operations are optional, the propagation will infer that only the first two operations can be present and will compute the exact conditional minimal start and end times for the two possibly present operations. We are not aware of any other framework that is capable of inferring such type of information on purely optional activities.

Most of the classical algorithms for propagating on STNs can be extended to handle conditional time-points. In CP Optimizer, the initial propagation of the temporal network is performed by an improved version of the Bellman-Ford

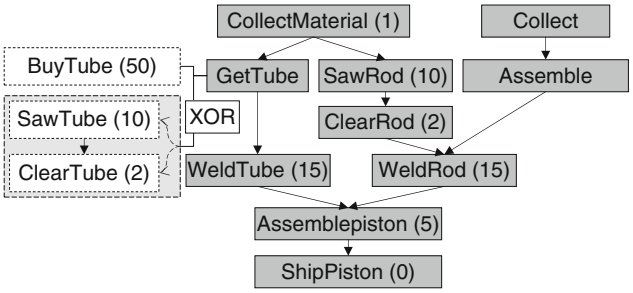


Fig. 6.9 Example of propagation

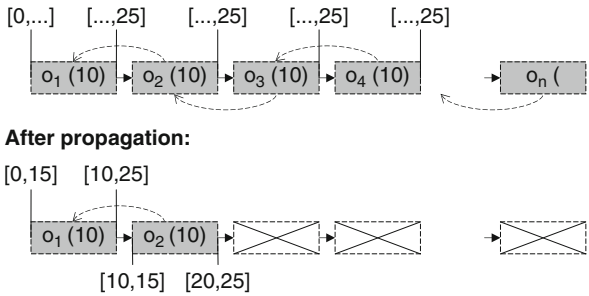


Fig. 6.10 Example of propagation

algorithm presented in [7] and the incremental propagation when a time-bound has changed or when a new arc or a new implication relation is detected is performed by an extension of the algorithm for positive cycles detection proposed in [6]. The main difference with the original algorithms is that propagation of the temporal bounds is performed only following those arcs that are allowed to propagate on their target given the implication relations. This propagation allows for instance to infer that a set of optional intervals with equivalent status forming a positive cycle in the temporal network cannot be present.

6.8.4 Interval Composition Constraints

The propagation of both the *span* and *alternative* constraints follow the same pattern. First, the part of the propagation that can be delegated to the logical and temporal networks is transferred to them:

- In the case of a span constraint span($a_0, \{a_1, ..., a_n\}$), the set of implications presenceOf(a_i) $\Rightarrow$ presenceOf(a_0) are treated by the logical network and the set of arcs *startsBeforeStart*(a_0, a_i) and *endsBeforeEnd*(a_i, a_0) by the temporal network.
- In the case of an alternative constraint alternative($a_0, \{a_1, ..., a_n\}$), the set of implications presenceOf(a_i) $\Rightarrow$ presenceOf(a_0) are treated by the logical network and the set of arcs *startsAtStart*(a_0, a_i) and *endsAtEnd*(a_i, a_0) by the temporal network.

The rest of the propagation is performed by the specific constraints themselves:

- The span constraint propagates the fact that when a_0 is present, there must exist at least one present interval variable a_i that starts at the same date as the start of a_0 (and the symmetrical relation for the end).
- The alternative constraint propagates the fact that no two interval variables a_i and a_j can simultaneously execute and maintains the conditional temporal bounds of interval variable a_0 as the constructive disjunction of the conditional temporal bounds of possibly present interval variables a_i. Propagation events on interval variables allow writing efficient incremental propagation for the alternative constraint.

6.8.5 Other Constraints

Classical constraint propagation algorithms have been extended to be able to handle optional interval variables and additional constraints. These algorithms include: time-tabling, precedence graphs or disjunctive constraint [1] and edge-finding variants [20]. For instance, for cumul functions, time-tabling algorithms have been extended to handle alwaysIn constraints on interval variables. For state functions, the time-tabling and disjunctive algorithms have been extended to handle the

Table 6.2 Constraint propagation algorithms

Model element	Inference level	Filtering algorithms
Sequence variable	**Basic**	Light precedence graph
	$\geq$ Medium	Precedence graph
No-overlap constraint	**Basic**	Timetable
	Medium	+ Disjunctive
	Extended	+ EF variants
Cumul function expression	**Basic**	Timetable
	Medium	+ Disjunctive
	Extended	+ EF variants
State function variable	**Basic**	Timetable
	$\geq$ Medium	+ Disjunctive

alignment specifications and the various types of incompatibilities between the alwaysIn, alwaysConstant, alwaysEqual and alwaysNoState constraints.

By default, light propagation algorithms with an average linear complexity are used. A set of inference level parameters is available to the user to perform additional filtering as summarized on Table 6.2.

6.8.6 Search

CP Optimizer implements a robust search algorithm to support the formalism described in this paper. This search was tested on an extensive library of models. It is based on the Self-Adapting Large Neighborhood Search described in [11] that consists of an improvement method that iteratively *unfreezes* and *re-optimizes* a selected fragment of the current solution.

Unfreezing a fragment relies on the notion of Partial Order Schedule [17]. This notion has been generalized to the modeling elements presented in this paper (no-overlap constraint, cumul function expressions and state variables).

The re-optimization of a partially unfrozen solution relies on a tree search using constraint propagation techniques.

6.9 Examples

6.9.1 Flow-Shop with Earliness and Tardiness Costs

6.9.1.1 Problem Description

The first problem studied in the paper is a flow-shop scheduling problem with earliness and tardiness costs on a set of instances provided by Morton and Pentico [15] that have been used in a number of studies including GAs [19] and Large Neighbourhood Search [8]. In this problem, a set of n jobs is to be executed on

a set of m machines. Each job i is a chain of exactly m operations, one per machine. All jobs require the machines in the same order that is, the position of an operation in the job determines the machine it will be executed on. Each operation j of a job i is specified by an integer processing time $pt_{i,j}$. Operations cannot be interrupted and each machine can process only one operation at a time. The objective function is to minimize the total earliness/tardiness cost. Typically, this objective might arise in just-in-time inventory management: a late job has negative consequence on customer satisfaction and time to market, while an early job increases storage costs. Each job i is characterized by its release date rd_i, its due date dd_i and its weight w_i. The first operation of job i cannot start before the release date rd_i. Let C_i be the completion date of the last operation of job i. The earliness/tardiness cost incurred by job i is $w_i \cdot abs(C_i - dd_i)$. In the instances of Morton and Pentico, the total earliness/tardiness cost is normalized by the sum of operation processing times so the global cost to minimize is:

$$\frac{\sum_{i\in[1,n]}\left(w_i \cdot abs(C_i - dd_i)\right)}{W} \text{ where } W = \sum_{i\in[1,n]}\left(w_i \cdot \sum_{j\in[1,m]} pt_{i,j}\right)$$

6.9.1.2 Model

A complete OPL model for this problem is shown in Model 1. The instruction at *line 1* tells the model is a CP model to be solved by CP Optimizer. The section between

Model 1 - OPL Model for Flow-shop with Earliness and Tardiness Costs

```
 1: using CP;
 2: int n = ...;
 3: int m = ...;
 4: int rd[1..n] = ...;
 5: int dd[1..n] = ...;
 6: float w[1..n] = ...;
 7: int pt[1..n][1..m] = ...;
 8: float W = sum(i in 1..n) (w[i] * sum(j in 1..m) pt[i][j]);
 9: dvar interval op[i in 1..n][j in 1..m] size pt[i][j];
10: dexpr int C[i in 1..n] = endOf(op[i][m]);
11: minimize sum(i in 1..n) w[i]*abs(C[i]-dd[i])/W;
12: subject to {
13:    forall(i in 1..n) {
14:       rd[i] <= startOf(op[i][1]);
15:       forall(j in 1..m-1)
16:          endBeforeStart(op[i][j],op[i][j+1]);
17:    }
18:    forall(j in 1..m)
19:       noOverlap(all(i in 1..n) op[i][j]);
20: }
```

line 2 and *line 8* is data reading and data manipulation. The number of jobs n is read from the data file at *line 2* and the number of machines m at *line 3*. A number of arrays are defined to store, for each on the n jobs, the release date (*line 4*), due date (*line 5*), earliness/tardiness cost weight (*line 6*) and, for each machine, the processing time of each operation on the machine (*line 7*). The normalization factor W is computed at *line 8*. The model itself is declared between *line 9* and *line 20*. *Line 9* creates a 2-dimensional array of interval variables indexed by the job i and the machine j. Each interval variable represents an operation and is specified with a size corresponding to the operation's processing time. *Line 10* creates one integer expression $C[i]$ for each job i equal to the end of the m^{th} (last) operation of the job. These expressions are used in *line 11* to state the objective function. The constraints are defined between *line 13* and *line 19*. For each job, *line 14* specifies that the first operation of job i cannot start before the job release date whereas precedence constraints between operations of job i are defined at *lines 15-16*. *Lines 18-19* state that for each machine j, the set of operations requiring machine j do not overlap.

6.9.1.3 Experimental Results

Table 6.3 compares the results obtained by the default automatic search of CP Optimizer using the above model (col. CPO) with the best results obtained by various genetic algorithms as reported in [19] (col. GA-best) and the results of the best Large Neighbourhood Search (S-LNS) studied in [8] (col. S-LNS-best). A time limit of one hour was used on a 3GHz processor for CP Optimizer similar to the two hours limit used in [8] on a 1.5GHz processor. The average improvement (using the geometric mean over the ratio $value_{CPO}/value_{Other}$) over the best GA is about 25% whereas the average improvement over the best LNS is more modest (1.7%).

6.9.2 *Satellite Scheduling*

6.9.2.1 Problem Description

The second illustrative model is an oversubscribed scheduling problem described in [10]. This model is a generalization of two real-world oversubscribed scheduling domains, the USAF Satellite Control Network (AFSCN) scheduling problem and

Table 6.3 Results for flow-shop scheduling with earliness and tardiness costs

Problem	GA-best	S-LNS-best	*CPO*	Problem	GA-best	S-LNS-best	*CPO*
jb1	0.474	0.191	*0.191*	ljb1	0.279	0.215	*0.215*
jb2	0.499	0.137	*0.137*	ljb2	0.598	0.508	*0.509*
jb4	0.619	0.568	*0.568*	ljb7	0.246	0.110	*0.137*
jb9	0.369	0.333	*0.334*	ljb9	0.739	1.015	*0.744*
jb11	0.262	0.213	*0.213*	ljb10	0.512	0.525	*0.549*
jb12	0.246	0.190	*0.190*	ljb12	0.399	0.605	*0.518*

the USAF Air Mobility Command (AMC) airlift scheduling problem. These two domains share a common core problem structure:

- A problem instance consists of n tasks. In AFSCN, the tasks are communication requests; in AMC they are mission requests.
- A set Res of resources are available for assignment to tasks. Each resource $r \in Res$ has a finite capacity $cap_r \geq 1$. The resources are air wings for AMC and ground stations for AFSCN. The capacity in AMC corresponds to the number of aircraft for a wing; in AFSCN it represents the number of antennas available at the ground station.
- Each task T_i has an associated set Res_i of feasible resources, any of which can be assigned to carry out T_i. Any given task T_i requires 1 unit of capacity (i.e., one aircraft in AMC or one antenna in AFSCN) of the resource $r_j \in Res_i$ that is assigned to perform it. The duration $Dur_{i,j}$ of task T_i depends on the allocated resource r_j.
- Each of the feasible alternative resources $r_j \in Res_i$ specified for a task T_i defines a time window within which the duration of the task needs to be allocated. This time window corresponds to satellite visibility in AFSCN and mission requirements for AMC.
- All tasks are optional; the objective is to minimize the number of unassigned tasks.[4]

6.9.2.2 Model

A complete OPL model for this problem is shown in Model 2 using the AFSCN semantics. The section between *line 2* and *line 6* is data reading and data

Model 2 - OPL Model for Satellite Scheduling

```
 1: using CP;
 2: tuple Station { string name; key int id; int cap; }
 3: tuple Alternative { string task; int station; int smin; int dur; int emax; }
 4: {Station} Stations = ...;
 5: {Alternative} Alternatives = ...;
 6: {string} Tasks = { a.task | a in Alternatives };
 7: dvar interval task[t in Tasks] optional;
 8: dvar interval alt[a in Alternatives] optional in a.smin..a.emax size a.dur;
 9: maximize sum(t in Tasks) presenceOf(task[t]);
10: subject to {
11:    forall(t in Tasks)
12:       alternative(task[t], all(a in Alternatives: a.task==t) alt[a]);
13:    forall(s in Stations)
14:       sum(a in Alternatives: a.station==s.id) pulse(alt[a],1) <= s.cap;
15: }
```

[4] A second type of model with task priorities is also described in [10]. In the present paper, we focus on the version without task priorities.

manipulation. A tuple defining ground stations data (with a name, a unique integer identifier and a capacity) is defined at *line 2* and read from the data file at *line 4*. A tuple defining a possible resource assignment for a task (specifying a task, a station, a task minimal start time, a task duration and a task maximal end time) is defined at *line 10* and read from the data file at *line 5*. The set of all tasks $Tasks$ is computed at *line 6* as the set of tasks used in at least one possible assignments.

Variables and constraints are defined between *line 7* and *line 15*. *Line 7* defines an array of interval variables indexed by the set of tasks *Tasks*. As tasks are optional and may be left unassigned, each of these interval variable is declared optional so that it can be ignored in the solution schedule. Each of the possible task assignments is defined as an optional interval variable in *line 8*. When present, these interval variables will be of size *dur* and belong to the time window $[smin, emax]$ of the assignment. This is expressed by the `size` and `in` OPL keywords in the interval variable declaration. The objective function is to maximize the number of assigned tasks, that is, the number of present tasks in the schedule; this is specified by a sum of presence constraints at *line 9*.

The constraints *lines 11–12* state that each task, if present, is the alternative among the set of possible assignments for this task, this is modeled by an alternative constraint: if interval $task[t]$ is present, then one and only one of the intervals $alt[a]$ representing a ground station assignment for $task[t]$ will be present and $task[t]$ will start and end together with this selected interval. As specified by the semantics of the alternative constraint, if the task is absent, then all the possible assignments related with this task are absent too. The limited capacity (number of antennas) of ground stations is modeled by *lines 13-14*. For each ground station s, a cumul function is created that represents the time evolution of the number of antennas used by the present assignments on this station s. This is a sum of unit pulse functions $\text{pulse}(alt[a], 1)$. Note that when the assignment $alt[a]$ is absent, the resulting pulse function is the zero function so it does not impact the sum. The resulting sum is constrained to be lower than the maximal capacity cap of the station. An interesting feature of the CP Optimizer model is that it handles optional tasks in a very transparent way: here, the fact that tasks are optional only impacts the declaration of task intervals at *line 7*. The notion of optional interval variable and the handling of absent intervals by the constraints and expressions of the model (here the alternative constraint and the cumul function expressions) allows an elegant modeling of scheduling problems involving optional activities and, more generally, optional and/or alternative tasks, recipes or modes.

6.9.2.3 Experimental Results

Table 6.4 compares the results obtained by the default automatic search of CP Optimizer using the above model (col. CPO) with the TaskSwap (TS) and Squeaky Wheel Optimization (SWO) approaches studied in [10] (col. TS and SWO). Figures represent the average number of unscheduled tasks for each problem set of the benchmark. The time limit for each instance was fixed to 120s for problem

Table 6.4 Results for satellite scheduling

Problem set	TS	SWO	*CPO*	Problem set	TS	SWO	*CPO*
1.1	30.44	26.60	*27.50*	4.1	3.20	2.00	*1.96*
1.2	114.02	104.72	*98.10*	4.2	13.34	7.90	*7.48*
1.3	87.92	84.52	*86.04*	4.3	16.60	12.46	*9.68*
2.1	11.46	7.80	*7.84*	5.1	3.90	3.80	*3.76*
2.2	45.54	34.26	*30.64*	5.2	32.98	31.98	*31.72*
2.3	33.96	31.18	*32.14*	5.3	46.18	45.22	*44.34*
3.1	2.64	2.32	*2.28*	6.1	1.56	1.28	*1.24*
3.2	15.50	12.82	*11.82*	6.2	11.62	9.56	*8.92*
3.3	32.10	28.58	*24.00*	6.3	25.28	22.60	*19.48*

sets $x.1$, 180s for problem sets $x.2$ and 360s for problem sets $x.3$. In average, compared to the best approach described in [10] (SWO), the default automatic search of CP Optimizer assigns 5.3% more tasks.

6.9.3 Personal Task Scheduling

6.9.3.1 Problem Description

The third problem treated in this paper is the personal task scheduling problem introduced in [18] and available as an add-on to Google Calendar (selfplanner.uom.gr/). It consists of a set of n tasks $\{T_1, ..., T_n\}$. Each task T_i has a duration denoted dur_i. All tasks are considered preemptive, i.e. they can be split into parts that can be scheduled separately. The decision variable p_i denotes the number of parts in which the i^{th} task has been split, where $p_i \geq 1$. T_{ij} denotes the j^{th} part of task T_i, $1 \leq j \leq p_i$. The sum of the durations of all parts of a task T_i must equal its total duration dur_i. For each task T_i, a minimum and maximum allowed duration for its parts, $smin_i$ and $smax_i$, as well as a minimum allowed temporal distance between every pair of its parts, $dmin_i$ are given. Depending on the values of $smax_i$ and $smin_i$ and the overall duration of the task dur_i, implicit constraints are imposed on p_i. For example, if $dur_i < 2 * smin_i$, then necessarily $p_i = 1$ and task T_i is non-preemptive. Each task T_i is associated with a domain $D_i = [s_{i1}, e_{i1}] \cup [s_{i2}, e_{i2}] \cup \cdots \cup [s_{iF_i}, e_{iF_i}]$, consisting of a set of F_i time windows within which all of its parts have to be scheduled. We denote respectively $L_i = s_{i1}$ and $R_i = e_{iF_i}$ the leftmost and rightmost values of domain D_i. A set of m locations is given, $Loc = \{l_1, l_2, \cdots, l_m\}$ as well as a 2-dimensional matrix $Dist$ with their temporal distances represented as non-negative integers. Each task T_i has its own spatial reference, $loc_i \in Loc$, denoting the location where the user should be in order to execute each part of the task. A set of ordering constraints, denoted $\prec (T_i, T_j)$ between some pairs of tasks is also defined, meaning that no part of task T_j can start its execution until all parts of task T_i have finished their execution. Time preferences are expressed for

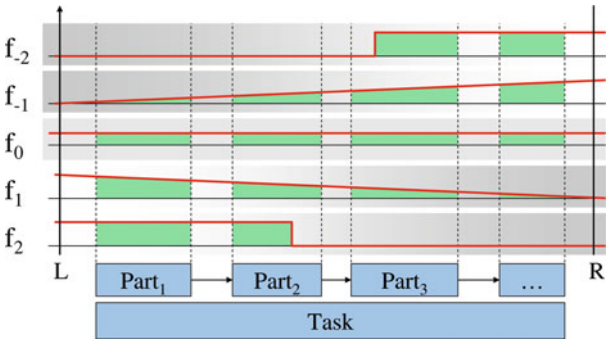


Fig. 6.11 Preference functions

each task T_i. Five types of preference functions are available; they are depicted on Fig. 6.11:

f_{-2} Execute as much as possible of task T_i after a date d.
f_{-1} Execute as much as possible of task T_i as late as possible.
f_0 No preference.
f_1 Execute as much as possible of task T_i as early as possible.
f_2 Execute as much as possible of task T_i before a date d.

For a given preference function f_i associated with a task T_i that is split into p_i parts $P_{i,1}, ..., P_{i,p_i}$, the satisfaction related with the execution of task T_i is computed as:

$$\text{satisfaction}(T_i) = \sum_{j=1}^{p_i} \sum_{t \in P_{i,j}} f_i(t)$$

It is to be noted that functions f_i are normalized in the interval [0,1] in such a way that an upper bound for the satisfaction for a task T_i is 1.

6.9.3.2 Model

A complete OPL model for the personal task scheduling problem is shown in Model 3. The section between *line 2* and *line 16* is data reading and data manipulation. A tuple representing a task description is declared at *line 2*, it specifies a unique integer task identifier, the location of the task, the task duration, the minimal and maximal duration of task parts, the minimal delay between two consecutive task parts, an identifier of the type of preference function for the task in $\{-2, -1, 0, 1, 2\}$, the threshold date in case preference function is of type f_{-2} or f_2 and two sets of integers ds and de respectively representing the start and end dates of the intervals $[s_i, e_i]$ of the task domain. The set of tasks is read from the data file at *line 3*. A triplet representing the temporal distance between two locations is declared at

Model 3 - OPL Model for Personal Task Scheduling

```
 1: using CP;
 2: tuple Task { key int id; int loc; int dur; int smin; int smax; int dmin; int f; int date; {int} ds;
    {int} de; }
 3: {Task} Tasks = ...;
 4: tuple Distance { int loc1; int loc2; int dist; };
 5: {Distance} Dist = ...;
 6: tuple Ordering { int pred; int succ; };
 7: {Ordering} Orderings = ...;
 8: int L[t in Tasks] = min(x in t.ds) x;
 9: int R[t in Tasks] = max(x in t.de) x;
10: int S[t in Tasks] = R[t]-L[t];
11: tuple Part { Task task; int id; }
12: {Part} Parts = { <t,i> | t in Tasks, i in 1 .. t.dur div t.smin };
13: tuple Step { int x; int y; }
14: sorted {Step} Steps[t in Tasks] =
15:   {<x,0> | x in t.ds} union {<x,1> | x in t.de};
16: stepFunction holes[t in Tasks] = stepwise(s in Steps[t]) {s.y -> s.x; 0};
17: dvar interval tasks[t in Tasks] in 0..500;
18: dvar interval a[p in Parts] optional size p.task.smin..p.task.smax;
19: dvar sequence seq in all(p in Parts) a[p] types all(p in Parts) p.task.loc;
20: dexpr float satisfaction[t in Tasks] = (t.f==0)? 1 :
21:   (1/t.dur)* sum(p in Parts: p.task==t)
22:     (t.f==-2)? maxl(endOf(a[p]),t.date)-maxl(startOf(a[p]),t.date) :
23:     (t.f==-1)? lengthOf(a[p])*(R[t]-(startOf(a[p])+endOf(a[p])-1)/2)/S[t] :
24:     (t.f== 1)? lengthOf(a[p])*((startOf(a[p])+endOf(a[p])-1)/2-L[t])/S[t] :
25:     (t.f== 2)? minl(endOf(a[p]),t.date)-minl(startOf(a[p]),t.date) : 0;
26: maximize sum(t in Tasks) satisfaction[t];
27: subject to {
28:   forall(p in Parts) {
29:     forbidExtent(a[p], holes[p.task]);
30:     forall(s in Parts: s.task==p.task && s.id==p.id+1) {
31:       endBeforeStart(a[p], a[s], p.task.dmin);
32:       presenceOf(a[s]) => presenceOf(a[p]);
33:     }
34:   }
35:   forall(t in Tasks) {
36:     t.dur == sum(p in Parts: p.task==t) sizeOf(a[p]);
37:     span(tasks[t], all(p in Parts: p.task==t) a[p]);
38:   }
39:   forall(o in Orderings)
40:     endBeforeStart(tasks[<o.pred>], tasks[<o.succ>]);
41:   noOverlap(seq, Dist);
42: }
```

line 4 and the transition distance matrix represented as a set of such triplets is read from the data file at *line 5*. A tuple storing an ordering constraint is defined on *line 6* and a set of such tuples is read from the data at *line 7*. *Lines 8–10* respectively compute, for each task t the leftmost value, rightmost value and diameter of the task domain. A tuple representing the ith part of a task is defined at *line 11* and the total

set of possible parts is computed at *line 12* considering that for each task of duration *dur* and minimal part duration *smin*, the maximal number of parts is $\lfloor dur/smin \rfloor$. *Lines 13–16* define a step function $holes[t]$ for each task t that is equal to 1 in the domain of t and to 0 everywhere else.

Variables and constraints are defined between *lines 17 and 42*. An array of interval variables, one interval $task[t]$ for each task t, is declared at *line 17*; each task is constrained to end before the schedule horizon (500 in the benchmark). *Line 18* defines an optional interval variable for each possible task part with a minimal and a maximal size given by $smin$ and $smax$. A sequence variable is created at *line 19* on the set of all parts p, each part being associated with an integer type in the sequence corresponding to the location of the part. The satisfaction expression for each task t is modeled on *lines 20-25* depending on the preference function type; it uses the OPL conditional expression `c?e1:e2` where c is a boolean condition and $e1$ is the returned expression if c is true and $e2$ the returned expression if c is false. The normalization factors are the ones used in [18].[5] The objective function, as defined on *line 26* is to maximize the sum of all tasks satisfaction.

The constraints on *line 29* forbid any part of a task t to overlap a point where the step function $holes[t]$ is zero; this will constrain each task part to be executed in its domain. Constraints on *lines 31-32* state that the set of parts of a given task t forms a chain of optional intervals with minimum separation time $dmin$ among which only the first ones will be executed, that is, each part $a[p]$ if present is constrained to be executed before its successor part $a[s]$ and the presence of part $a[s]$ implies the presence of part $a[p]$. Constraints on *line 36* state that the total duration of the part of a task must equal the specified task duration dur. Note that when part $a[p]$ is absent, by default the value of sizeOf($a[p]$) is 0. *Line 37* constrains each task t to span its parts, that is to start at the start of first part and to end with the end of the last executed part. Ordering constraints are declared on *line 40* whereas *line 41* states that task parts cannot overlap and that they must satisfy the minimal transition distance between task locations defined by the set of triplets *Dist*.

6.9.3.3 Experimental Results

Table 6.5 compares the results obtained by the default automatic search of CP Optimizer using the above model (col. CPO) and a time limit of 60s for each problem with the Squeaky Wheel Optimization (SWO) approach implemented in SelfPlanner [18] (col. SWO). CP Optimizer finds a solution to more problems than the approach described in [18]: the SWO could not find any solution for the problems with 55 tasks whereas the automatic search of CP Optimizer solves 70%

[5]The objective expression being quite complex, we used the solution checker provided with the instances to check that the constraints and objective function of our model are equivalent to the ones used in [18].

Table 6.5 Results for personal task scheduling

#	SWO	*CPO*	#	SWO	*CPO*	#	SWO	*CPO*	#	SWO	*CPO*
15-1	12.95	*14.66*	30-6	28.09	*29.28*	40-1	24.72	*28.95*	45-6	32.70	*37.35*
15-2	12.25	*13.16*	30-7	23.80	*24.20*	40-2	23.48	*32.07*	45-7	32.40	*35.77*
15-3	13.71	*13.90*	30-8	24.06	*26.89*	40-3	33.57	*37.74*	45-8	31.79	*35.23*
15-4	11.57	*12.55*	30-9	23.42	*24.86*	40-4	31.46	*35.45*	45-9	35.79	*38.86*
15-5	12.64	*14.67*	30-10	22.04	*27.18*	40-5	28.05	*34.21*	45-10	32.78	*40.68*
15-6	14.30	*14.63*	35-1	28.80	*31.56*	40-6	29.46	*34.01*	50-1	42.04	*43.53*
15-7	13.08	*14.46*	35-2	29.17	*32.33*	40-7	33.13	*37.51*	50-2	×	×
15-8	11.46	*12.37*	35-3	27.84	*28.58*	40-8	29.72	*34.90*	50-3	×	*37.17*
15-9	11.44	*11.61*	35-4	26.64	*29.67*	40-9	33.03	*36.89*	50-4	×	*36.52*
15-10	12.07	*13.51*	35-5	25.15	*32.13*	40-10	30.28	*34.19*	50-5	34.25	*43.55*
30-1	24.17	*29.13*	35-6	26.12	*29.49*	45-1	37.42	*42.90*	50-6	38.32	*41.87*
30-2	24.69	*27.55*	35-7	29.28	*31.69*	45-2	33.97	*39.71*	50-7	32.59	*42.48*
30-3	25.61	*26.53*	35-8	25.71	*30.07*	45-3	35.44	*39.40*	50-8	34.70	*43.67*
30-4	27.13	*28.49*	35-9	23.74	*29.60*	45-4	33.02	*37.41*	50-9	×	*42.75*
30-5	23.89	*26.46*	35-10	30.70	*33.41*	45-5	30.83	*36.65*	50-10	37.46	*41.84*
55-1	×	*36.84*	55-4	×	*40.36*	55-7	×	×	55-10	×	×
55-2	×	*38.56*	55-5	×	*42.70*	55-8	×	*45.27*			
55-3	×	×	55-6	×	*35.92*	55-9	×	*42.14*			

of them. Furthermore, SWO could not find any solution to 4 of the smaller problems with 50 tasks whereas CP Optimizer solves them all but for problem 50-2. On problems where SWO finds a solution, the average task satisfaction (average of the ratio between the total satisfaction and the number of tasks) is 78% whereas it is 87.8% with CP Optimizer. It represents an improvement of about 12.5% in solution quality.

6.10 Conclusion

The algebraic model presented in this paper has been implemented in CP Optimizer and is available in C++, Java, C# as well as in OPL [12, 13]. In complement of the notion of *optional interval variable* that considerably simplifies the modeling of complex scheduling structures (optional activities, alternative modes or recipes), a set of global variables and expressions are used for each aspect of a scheduling problem: *sequence variables* for interval sequencing, *cumul function expressions* for cumulative reasoning and *state function variables* for representing the time evolution of a state variable. A powerful set of constraints on these variables and expressions is provided. As all these constraints handle the optional status of interval variables, they can be posted even on optional or alternative parts of the schedule to effectively prune part of the search space.

The clear separation between (1) the structure of scheduling problems captured with composition constraints on optional interval variables such as *span* and

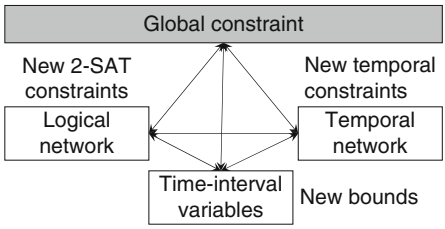


Fig. 6.12 General approach for propagation

alternative and (2) the resource constraints expressed as mathematical concepts such as sequences or functions results in very simple, elegant and concise models. For instance, the model for the classical Multi-Mode Resource-Constrained Project Scheduling Problem with both renewable and non-renewable resources is less than 50 lines long in OPL including data manipulation.

The automatic search algorithm has shown to be robust and efficient for solving a large panel of models as shown in [11].

As illustrated on Fig. 6.12, coupling global constraints with logical and temporal network information allows performing stronger conjunctive deductions on the bounds of interval variables and inferring new logical and temporal relations. Future work will consist in enhancing the current global constraints following this general pattern.

References

1. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-Based Scheduling. Applying Constraint Programming to Scheduling Problems. Kluwer Academics (2001)
2. Barták, R., Čepek, O.: Temporal networks with alternatives: Complexity and model. In: Proceedings of FLAIRS-2007 (2007)
3. Beck, J.C., Fox, M.S.: Scheduling alternative activities. In: Proceedings of AAAI-99 (1999)
4. Bessière, C., Régin, J.C.: Arc consistency for general constraint networks: preliminary results. In: Proceedings of IJCAI'97, pp. 398–404 (1997)
5. Brafman, R.I.: A simplifier for propositional formulas with many binary clauses. In: Proceedings of IJCAI-01, pp. 515–522 (2001)
6. Cesta, A., Oddi, A.: Gaining efficiency and flexibility in the simple temporal problem. In: Proceedings of TIME-96 (1996)
7. Cherkassky, B., Goldberg, A., Radzic, T.: Shortest paths algorithms: Theory and experimental evaluation. Math. Program. **73**, 129–174 (1996)
8. Danna, E., Perron, L.: Structured vs. unstructured large neighborhood search: A case study on job-shop scheduling problems with earliness and tardiness costs. In: Proceedings of CP 2003, pp. 817–821 (2003)
9. ILOG: ILOG CP Optimizer (2008); URL Web-page:http://www.ilog.com/products/cpoptimizer/. Http://www.ilog.com/products/cpoptimizer/
10. Kramer, L.A., Barbulescu, L.V., Smith, S.F.: Understanding Performance Tradeoffs in Algorithms for Solving Oversubscribed Scheduling. In: Proceedings of 22nd AAAI Conference on Artificial Intelligence (AAAI-07), pp. 1019–1024 (2007)
11. Laborie, P., Godard, D.: Self-Adapting Large Neighborhood Search: Application to single-mode scheduling problems. In: Proceedings of MISTA-07 (2007)

12. Laborie, P., Rogerie, J.: Reasoning with Conditional Time-intervals. In: Proceedings of FLAIRS-08 (2008)
13. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: Reasoning with Conditional Time-intervals, Part II: an Algebraical Model for Resources. In: Proceedings of 22th International FLAIRS Conference (FLAIRS 2009) (2009)
14. Moffitt, M.D., Peintner, B., Pollack, M.E.: Augmenting disjunctive temporal problems with finite-domain constraints. In: Proceedings of AAAI-2005 (2005)
15. Morton, T., Pentico, D.: Heuristic Scheduling Systems. Wiley, New York (1993)
16. Nuijten, W., Bousonville, T., Focacci, F., Godard, D., Le Pape, C.: Towards an industrial manufacturing scheduling problem and test bed. In: Proceedings of PMS-2004, pp. 162–165 (2004)
17. Policella, N., Cesta, A., Oddi, A., Smith, S.: Generating robust schedules through temporal flexibility. In: Proceedings of ICAPS 04. Whistler, Canada (2004)
18. Refanidis, I.: Managing personal tasks with time constraints and preferences. In: Proceedings of 17th International Conference on Automated Planning and Scheduling Systems (ICAPS-07) (2007)
19. Vázquez, M., Whitley, L.D.: A comparison of genetic algorithms for the dynamic job shop scheduling problem. In: Proceedings of GECCO 2000 (2000); URL citeseer.ist.psu.edu/528131.html
20. Vilím, P.: Global constraints in scheduling. Ph.D. thesis, Charles University in Prague, Faculty of Mathematics and Physics (2007)

Chapter 7
Using ZIMPL for Modeling Production Planning Problems

Ulrich Dorndorf, Stefan Droste, and Thorsten Koch

Abstract In this chapter we describe, how ZIMPL is used as an algebraic modeling language for mixed-integer linear programs (MIPs) at the INFORM GmbH (Aachen, Germany). We firstly give an overview of ZIMPL, thereby motivating the choice of ZIMPL for modeling a production planning problem. Besides depicting the role of ZIMPL during the development and training process of optimization software at INFORM, we show explicitly how a basic model of a production planning problem, the MLCLSP, can be modeled with ZIMPL, thereby demonstrating its ease-of-use and some of its features.

7.1 Introduction

The development of optimization software includes as one very essential step the choice of the concrete optimization algorithm. If the problem at hand is a well-known optimization problem (or only a slight variation of it), chances are high that a specialized efficient algorithm can be used to solve it. As soon as an efficient algorithm for the problem is not known (e.g., because the problem is NP-complete) or the problem is likely to be modified often according to customer's wishes, a more general and flexible algorithmic approach and problem formulation is needed.

In this situation modeling the problem with an algebraic modeling language (AML) is a good option, since this allows one to easily modify the problem until it has found its final form. Many AMLs allow to formulate problems of different classes, e.g. linear, mixed-integer linear, quadratic, or convex programming, only

U. Dorndorf (✉) · S. Droste
INFORM GmbH, Pascalstraße 23, 52076 Aachen, Germany
e-mail: ulrich.dorndorf@inform-software.com; stefan.droste@inform-software.com

T. Koch
Zuse Institut Berlin, Takustraße 7, 14195 Berlin, Germany
e-mail: koch@zib.de

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_7, © Springer-Verlag Berlin Heidelberg 2012

some of which can be solved efficiently. Choosing the problem class early on is therefore important, since it has an influence on the problem size that can be realistically solved. It is important to note that the solvability of a MIP is hard to predict and may depend very much on the details of the modeling, see, e.g. [2].

Our choice of formulating the problem as a linear or mixed-integer linear program has many advantages:

- Linear programs are quite powerful, as many real-world restrictions and optimization goals can at least be approximately modeled by linear (in-)equalities.
- They are easily comprehensible even for people not trained in formulating linear programs.
- There exist a number of modern and efficient solvers, which can solve linear programs of surprisingly large size and even for mixed-integer linear programs with many integer variables give at least very good solutions.
- Furthermore, the dual bound of a linear program allows the user to upper bound the gap between the best found primal solution and the best possible solution, i. e. the user has knowledge about the maximal difference between the objective value of the current solution and the best possible solution.

All these advantages together with good experiences made in previous projects have led to linear programs as the modeling paradigm for a new optimization module in add*ONE, a general production planning system of INFORM allowing cost-optimal and resource-observing lot sizing. Of course, using linear programs when developing the best problem formulation can be tedious, because the linear program mixes the different restrictions of the problem with the data used for testing the problem formulation.

An AML allows a clear separation between the model and the data, making it ideal for the prototyping process, where the problem goals and constraints often change. Additionally, a formulation in an AML is much shorter and easier to understand than a linear program. The algebraic modeling language we chose was ZIMPL, because it:

- Is freely available.
- Has a very clear syntax close to the mathematical notation.
- Has very powerful features, and
- Is well integrated with SCIP [1], a MIP-solving package.

In the following we at first give an overview of ZIMPL, its goals and features. To exemplify some of these features, we show how the MLCLSP (multi-level capacitated lot sizing problem), a well-known basic model for our optimization module, can be formulated in ZIMPL. Finally we describe how ZIMPL was used in the development and training process of a new optimization module for INFORM's production planning system.

7.2 ZIMPL

ZIMPL (Zuse Institute Mathematical Programming Language) is a powerful language to describe mathematical programs and also a tool to translate these programs into LP-files, one of the standard descriptions of linear programs that can be solved with many solvers. In particular the design goals for the development of ZIMPL are to have a modeling language that:

- Is freely available with C source code under the LGPL.
- Is highly portable (Linux, Windows, MacOS-X, Solaris, ...).
- Is quick and easy to learn.
- Is solver independent.
- Is available as a library.
- Can be used standalone or linked to a solver.
- Can be used for lectures.
- Is stable enough for industry projects, and
- is using rational arithmetic to ensure numerical correctness of the generated model.

Beginning in the eighties algebraic modeling languages like GAMS [4, 7] and AMPL [9, 10] were developed. With these tools it became possible to state an LP in near mathematical notation and have it automatically translated into a file a solver can process or even directly fed into the appropriate solver.

Most modeling languages are commercial products. None of these languages is available as source code for further development. None can be given to colleagues or used in classes, apart from very limited "student editions". Usually only a limited number of operating systems and architectures is supported, sometimes only Microsoft Windows.

In contrast, ZIMPL is freely available in source code under LGPL. This is very convenient for teaching, as all students can use ZIMPL on their laptops and at home. It should be noted that the general situation has improved since 1999 when ZIMPL started. There are now at least some other open source languages, like, for example, the GNU MATHPROG language, which implements a subset of AMPL and is part of the GNU linear programming kit.[1]

The current trend in commercial modeling languages is to further integrate features like data base query tools, solvers, report generators, and graphical user interfaces. This sometimes even allows to build complete graphical (business) applications around the mathematical model. Today the freely available modeling languages are no match in this regard. ZIMPL implements maybe 20% of the functionality of AMPL. Nevertheless, having perhaps the most important 20% proved to be sufficient for many real-world projects. Furthermore, the user manual for ZIMPL requires only about 25 pages. As we will see in the following sections,

[1] http://www.gnu.org/software/glpk

you can learn ZIMPL within a few hours and in contrast to graphical modelers ZIMPL models are still very near to the mathematical notation.

According to [8], modeling languages can be separated into two classes, namely the *independent modeling languages*, which do not rely on a specific solver and the *solver modeling languages*, which are deeply integrated with a specific solver. In exchange for having a better integration with the solvers, it is no longer possible to easily switch between solvers as with independent languages. Given that the performance of a particular solver varies highly with the particular model used, see, e.g. [12], we believe that the ability to switch between different solvers is actually one of the more useful features of a modeling language.

ZIMPL is completely solver independent and can be called stand-alone, generating an LP-, MPS- or POL-file.[2] It also can be used as a callable library to provide a problem reader function for a specific solver. SCIP,[3] and lp_solve,[4] for example, can read ZIMPL models directly.

During the development of ZIMPL special focus has been placed on software engineering. ZIMPL comes together with a test suite that when executed covers more than 80% of the program code. This assures that changes and new features do not break existing functionality. In particular this includes all error and warning messages. Assert statements are used extensively in the code to test preconditions and invariants. Special targets in the Makefile are available for performing valgrind[5] checks and the code is regularly run through flexelint[6] a static program checker.

What makes ZIMPL special is the use of rational arithmetic. With a few noted exceptions all computations in ZIMPL are done with infinite precision rational arithmetic, ensuring that no rounding errors can occur during modeling. This is achieved by employing the GNU Multi Precision Library.[7]

Another special feature is an automatic conversion of functions with decision variables as arguments into a system of inequalities. The arguments of these functions have to be linear terms consisting of bounded integer or binary variables. For example, it is possible to have constraints dependent on the value of a variable or compute the absolute value of a variable.

ZIMPL also supports polynomial terms, but of course the solver used has to support this. In connection with SCIP it is now possible to solve non-convex quadratic integer programs [3].

In general each ZIMPL model consists of six types of statements:

- Sets
- Parameters
- Variables

[2] see http://polip.zib.de for the POL-format.

[3] http://scip.zib.de

[4] http://lpsolve.sourceforge.net

[5] http://valgrind.org

[6] http://www.gimpel.com

[7] http://gmplib.org

- Objective
- Constraints
- Function definitions

ZIMPL statements never change the already existing part of the model but only add to it. This makes it much easier to understand ZIMPL models.

ZIMPL has been (and is being) used to model many real-world and educational questions, as diverse as location planning in telecommunications, 3D-Steiner tree packing for chip design, track auctioning, protein folding, lectures at many universities, and courses like Combinatorial Optimization at Work.[8] ZIMPL can be downloaded as source or pre-compiled binary at http://zimpl.zib.de.

7.3 Formulating the MLCLSP with ZIMPL

To demonstrate ZIMPL's ease-of-use we will not give a complete overview of its feature set (see its manual at zimpl.zib.de instead), but give a rather short, but complete example of a problem formulation in ZIMPL. The Multi-Level Capacitated Lot-Sizing-Problem (MLCLSP) used for this example is a classical model of production planning (see e. g. [6] for the MLCLSP or [13] for a general overview of MIP-models for production planning) and the basis of the model used at INFORM. Since many of the additional features used in the final complete production model make the formulation harder to understand without exemplifying new features of ZIMPL, we restrict ourselves here to the MLCLSP.

The MLCLSP is a classical model for a production planning: Suppose we have a set P of products and a set T of time periods. For every pair $p, t \in P \times T$ of product and time period we want to determine a production quantity $x_{p,t} \geq 0$, stating how many units of p are produced in period t. Therefore, our model contains $|P \times T|$ variables $x_{p,t} \geq 0$.

To define a variable $x \geq 0$ in ZIMPL it is sufficient to write

```
var x >= 0;
```

Notice that it is equivalent to write

```
var x;
```

as variables have a default lower resp. upper bound of 0 resp. infinity. But adding the lower bound explicitly avoids ambiguities.

If the production quantity x can only take integer values, the definition has to be changed only slightly by:

```
var x integer >= 0;
```

To define variables $x_p \geq 0$ for all products $p \in P$, we first need to define the set P. Suppose P contains exactly the elements $P1$, $P2$, and $P3$:

[8] see http://co-at-work.zib.de

```
set P := {"P1","P2","P3"};
```

Now the variables $x_p \geq 0$ can be defined for all $p \in P$ simply by stating:

```
set P := {"P1","P2","P3"};
var x[P] >= 0;
```

Since we want to define the variables $x_{p,t} \in \mathbb{N}$ for all pairs $p, t \in P \times T$, we first have to define the sets P and T (assuming that T is $\{1, 2, \ldots, 12\}$, e. g. modeling the different months of a year):

```
set P := {"P1","P2","P3"};
set T := {1 to 12};
```

Then the 36 variables $x_{p,t}$ can be defined in ZIMPL by the single line:

```
var x[P*T] >= 0;
```

Even these very first examples show that ZIMPL allows a very simple and intuitive definition of sets and variables, where e. g. $P * T$ is the cartesian product of the sets P and T and {1 to 12} defines the set of all integers between 1 and 12.

To be a valid solution of the MLCLSP, a production plan, i. e. an assignment of the variables $x_{p,t}$, has to fulfil different constraints, modeling the real-world requirements one considers to be essential.

The first real-world requirement is that every production takes place on a set of resources (e. g. machines) having only limited capacity.

This is usually modeled by a constraint ensuring that production does not exceed the capacity $c_{r,t} \in \mathbb{N}$ of resource $r \in R$ in period $t \in T$. Therefore, we have to define a set R of resources (assuming that $R = \{R1, R2\}$):

```
set R := {"R1","R2"};
```

Additionally, we have to define the parameters $c_{r,t}$ (the capacity of resource r in period t) and $u_{p,r}$ (the capacity used of resource r by producing one unit of product p):

```
param c[R*T] :=
      |  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12 |
|"R1"| 20, 10, 10, 10, 20, 10, 30, 10, 10, 10, 10,  0 |
|"R2"| 50, 10, 20, 10, 10, 10,  0,  0,  0,  0,  0,  0 |;
```

It is obvious that this notation corresponds directly to a table containing all values of $c_{r,t}$, where the rows are indexed by the elements of R and the columns by those of T. The same has to be done by defining the values of $u_{p,r}$:

```
param u[P*R] :=
      |"R1", "R2"|
|"P1"|    1,    0 |
|"P2"|    0,    2 |
|"P3"|    1,    1 |;
```

Given these parameters, the constraint guaranteeing that the capacity of all resources is not exceeded by the production plan can be written as:

```
subto ResUse:
forall <r,t> in R*T: sum <p> in P: x[p,t]*u[p,r] <= c[r,t];
```

Notice how the ZIMPL-code directly corresponds to the mathematical notation of the constraint:

$$\forall r, t \in R \times T : \sum_{p \in P} x_{p,t} \cdot u_{p,r} \leq c_{r,t}.$$

This becomes even more obvious when comparing the ZIMPL-notation to the LaTeX-notation of the expression above:

```
\forall r,t \in R \times T:
\sum_{p \in P} x_{p,t} \cdot u_{p,r} \leq c_{r,t}.
```

Another important constraint of the MLCLSP has to ensure that in every period t the stock of product p is at least equal to the given external demand $d_{p,t}$ and diminished by this demand. The external demand is easily defined by a parameter table:

```
param d[P*T] :=
      |  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12 |
 |"P1"|  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 |
 |"P2"|  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 |
 |"P3"|  1,  1,  2,  0,  0,  1,  1,  2,  0,  2,  2,  2 |;
```

To model the stock amount of product p in time period t, we introduce new variables $s_{p,t}$ for all pairs of product $p \in P$ and time period $t \in T$:

```
var s[P*T union P*{0}] >= 0;
```

The observant reader will notice that the above line defines $s_{p,t}$ for all pairs of product $p \in P$ and time period $t \in T \cup \{0\}$, i. e. they are additionally defined for $t = 0$. This is motivated by the stock balance constraint (see below), where the stock amount in period $t - 1$ influences the amount in period t.

To model the stock amount, it is important that a specified number of products $p \in P$ can be necessary to produce a product $q \in P$, if p is a pre-product of q. This is usually modeled in the MLCLSP by coefficients $b_{p,q}$, indicating that $b_{p,q}$ units of p are necessary to produce one unit of q:

```
param b[P*P] :=
      | "P1", "P2", "P3" |
 |"P1"|    0,    0,    2 |
 |"P2"|    1,    0,    0 |
 |"P3"|    0,    0,    0 |;
```

In this example, product $P3$ is the sole end-product, $P1$ its only pre-product, and $P2$ the only pre-product of $P1$, since the row index specifies the pre-product and the column index the post-product. Hence, production of one unit of $P3$ needs two units of $P1$ and production of one unit of $P1$ needs one unit of $P2$.

Given those dependencies, the so-called stock balance equality guarantees that $s_{p,t}$ is always equal to the stock amount of product p in time period t, depending on the production quantities $x_{p,t}$ and the demands $d_{p,t}$:

$$s_{p,t-1} + x_{p,t} = d_{p,t} + \sum_{q \in P} b_{p,q} \cdot x_{q,t} + s_{p,t}.$$

The equality's validity can be easily checked by interpreting its left-hand side as the number of units "flowing into" the stock of p in period t: the stock $s_{p,t-1}$ of p at the end of period $t-1$ and the production quantity $x_{p,t}$ of p in period t. Analogously, the right-hand side contains the number of units "flowing out" of the stock: the external demand $d_{p,t}$ of p in period t, the (internal) demand $\sum_{q \in P} b_{p,q} \cdot x_{q,t}$ caused by the production of post-products q of p, and the stock $s_{p,t}$ of p at the end of period t.

This equality can be easily formulated in ZIMPL:

```
subto Balance:
forall <p,t> in P*T:
s[p,t-1]+x[p,t] == d[p,t] + sum <q> in P: b[p,q]*x[q,t] + s[p,t];
```

While on the first sight this equality only guarantees that the s-variables represent the stock amount, it also ensures that this stock is always sufficient to satisfy the demand, i. e. that for all products p and time periods t we have $s_{p,t-1} + x_{p,t} \geq d_{p,t}$. This is simply caused by the non-negativeness of the stock variables $s_{p,t}$: a demand $d_{p,t}$ larger than $s_{p,t-1} + x_{p,t}$ would imply a negative stock amount $s_{p,t}$ and is therefore not possible.

The balance equalities are essential for the MLCLSP, since they are responsible for $s_{p,t}$, modeling the stock amount and thereby "connecting" the variables of the different time periods. Therefore, it is important to fix the initial stock amount $s_{p,0}$ to 0 or to some other given value. Otherwise a solution could base on setting the $s_{p,0}$-variables to large values, while all production variables $x_{p,t}$ are zero. Setting these variables to 0 in ZIMPL is easy:

```
subto InitS: forall <p> in P: s[p,0] == 0;
```

Obviously, our linear program is missing something essential: the objective function to be optimized! In the case of the MLCLSP we want to minimize the costs of the production plan, i. e. the holding costs and the setup costs (the production costs are omitted, since they are inevitable in every production plan meeting the demand).

While the holding costs can be easily computed using the stock variables $s_{p,t}$, the setup costs need some additional effort. The idea behind the setup costs is to penalize a production event during a time period, e. g. modeling the effort to prepare machines for production. While the production variables $x_{p,t} \geq 0$ model the number of units of p produced in period t, setup costs are not dependent on the number of units produced, but only on the fact whether at least one unit is produced.

Hence, we need additional binary setup variables $y_{p,t}$, indicating whether product p is produced in period t or not:

```
var y[P*T] binary;
```

To guarantee that $y_{p,t} = 1$, if $x_{p,t} > 0$, we use the usual technique of a "Big M"-inequality, i. e. adding the constraint

$$x_{p,t} \leq M \cdot y_{p,t}$$

where M is a given parameter at least as large as any value of $x_{p,t}$. If the value of $x_{p,t}$ is positive, this inequality implies $y_{p,t} = 1$ (here it is important that $x_{p,t} \leq M$). Usually, M is chosen as a very large constant, e. g. $M := 999999$. Notice that it can be worthwhile to choose a much smaller value (e. g. by computing the Echelon demand, i. e. the maximal number of units of p that have to produced up to period t) to speed up the solving process.

The alert reader will notice that the above inequality does not enforce a setup variable $y_{p,t}$ to be zero, when $x_{p,t} = 0$. While this could easily be done by stating $y_{p,t} \leq x_{p,t}$, this is not necessary, if a positive value of a setup variable is penalized in the objective function.

To formulate the "Big M"-constraint in ZIMPL we first have to define the parameter M:

```
param M := 999999;
```

and then the BigM-constraint:

```
subto BigM: forall <p,t> in P*T: x[p,t] <= M * y[p,t];
```

Now we can formulate the objective function to be minimized, penalizing holding and setup costs:

$$\text{minimize} \sum_{p,t \in P \times T} \left(c^s_{p,t} \cdot y_{p,t} + c^h_{p,t} \cdot s_{p,t} \right),$$

where $c^s_{p,t}$ resp. $c^h_{p,t}$ are the positive parameters measuring the costs of setting up the resources for producing p in period t resp. holding one unit of p at the end of period t.

Analogously to defining the parameters $b_{p,q}$ and $u_{p,r}$, we have to formulate the parameters $c^s_{p,t}$ and $c^h_{p,t}$ prior to formulating the objective function:

```
param cs[P*T] :=
      |  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12 |
 |"P1"|  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5 |
 |"P2"|  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 |
 |"P3"|  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2 |;

param ch[P*T] :=
      |  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12 |
 |"P1"|  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5 |
 |"P2"|  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 |
 |"P3"| 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10 |;
```

Now, the objective function can be formulated by:

```
minimize cost: sum <p,t> in P*T: (cs[p,t]*y[p,t]+ch[p,t]*s[p,t]);
```

All in all, a complete formulation of the MLCLSP in ZIMPL looks as follows:

```
set P := {"P1","P2","P3"};
set T := {1 to 12};
set R := {"R1","R2"};
```

```
param d[P*T] :=
      |  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12 |
 |"P1"|  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 |
 |"P2"|  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 |
 |"P3"|  1,  1,  2,  0,  0,  1,  1,  2,  0,  2,  2,  2 |;

param c[R*T] :=
     |  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12 |
|"R1"| 20, 10, 10, 10, 20, 10, 30, 10, 10, 10, 10,  0 |
|"R2"| 50, 10, 20, 10, 10, 10,  0,  0,  0,  0,  0,  0 |;

param cs[P*T] :=
      |  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12 |
 |"P1"|  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5 |
 |"P2"|  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 |
 |"P3"|  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2 |;

param ch[P*T] :=
      |  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12 |
 |"P1"|  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5 |
 |"P2"|  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 |
 |"P3"| 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10 |;

param u[P*R] :=
     |"R1", "R2"|
|"P1"|   1,   0 |
|"P2"|   0,   2 |
|"P3"|   1,   1 |;

param b[P*P] :=
      | "P1", "P2", "P3" |
 |"P1"|    0,    0,    2 |
 |"P2"|    1,    0,    0 |
 |"P3"|    0,    0,    0 |;

param M := 999999;

var x[P*T] >= 0;
var s[P*T union P*{0}] >= 0;
var y[P*T] binary;

minimize cost: sum <p,t> in P*T: (cs[p,t]*y[p,t]+ch[p,t]*s[p,t]);

subto Balance: forall <p,t> in P*T:
s[p,t-1]+x[p,t] == d[p,t] + sum <q> in P: b[p,q]*x[q,t] + s[p,t];
subto ResUse:
forall <r,t> in R*T: sum <p> in P: x[p,t]*u[p,r] <= c[r,t];
subto BigM: forall <p,t> in P*T: x[p,t] <= M * y[p,t];
subto InitS: forall <p> in P: s[p,0] == 0;
```

While this example shows nicely how a classical model of operations research can be formulated with ZIMPL, its form is not optimal when integrating the model as part of a production planning system. Because in this situation the input data, e. g. the number of periods, the set of products resp. resources, or the different parameters

change frequently, making it necessary to also change the ZIMPL-model. Certainly, it would be much easier to separate model and data.

ZIMPL facilitates this separation with commands for parsing text files and interpreting its content as parameters. For example assume that the external demand $d_{p,t}$ is given in a separate text file in the following form:

```
#  Product  Period  Demand
d: P3       1       1
d: P3       2       1
d: P3       3       2
d: P3       6       1
d: P3       7       1
d: P3       8       2
d: P3       10      2
d: P3       11      2
d: P3       12      2
```

Then this file (named "MLCLSP.dat") can be read and its content assigned to the $d_{p,t}$-variables by the following statement:

```
param d[P*T] := read "MLCLSP.dat" as
                "<2s,3n> 4n" match "d:" comment "#" default 0;
```

Without going into details ZIMPL's read-command automatically splits a line, whenever a space, tab, or semicolon is read. Then the expression "2s" resp. "3n" represents the second resp. third entry, interpreted as a string resp. a number. While the expression in parentheses describes the coefficient's indices, the fourth entry stands for the coefficient's value. Hence, e. g. the line

```
d: P3       6       1
```

implies that $d_{P3,6} = 1$. All values $d_{p,t}$ for products p and periods t not declared in the file are set to the default value 0. Therefore, the above text file is equivalent to the former definition

```
param d[P*T] :=
      |  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12 |
 |"P1"| 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 |
 |"P2"| 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 |
 |"P3"| 1,  1,  2,  0,  0,  1,  1,  2,  0,  2,  2,  2 |;
```

To allow the definition of other parameters in the same file, the option "match" is essential: it filters exactly those lines, starting with the given string (in our case "d:"). Analogously, the option "comment" allows to define comment lines (in our case all those starting with "#").

While an explicit variable declaration like "d[P3,6]=1" in the text file may be easier to understand, the above format allows to easily define new parameters for all product-period pairs just by adding a column in the above text block.

Because of space restrictions we omit more details here, but it should be clear that all explicit value assignments of parameters in the ZIMPL-model can be replaced by separate data files and reading in their values with commands similar to the one shown above.

This technique was used in the development process at INFORM, as it allows a clean separation between the model (i.e. all variables, constraints and the objective function) and the data (i.e. the values of all parameters). Thereby the model could be extended or varied while the data can be changed in parallel.

With a given model and data set ZIMPL generates a standard description of the MIP as an LP- or MPS-file, which can be used as input for all standard MIP-solvers. Additionally, SCIP can directly read and process ZIMPL-files, making an explicit call of ZIMPL unnecessary.

7.4 ZIMPL in INFORM's Training and Workflow

As previously mentioned, ZIMPL was INFORM's choice of modeling language for a new production planning system especially because of its free availability and its close notational similarity to standard mathematical formulations. While the latter may be not important when experts in algebraic modeling language are responsible for the development of the optimization model, it becomes essential when modeling knowledge should be shared among many people.

From the very beginning this was a goal while developing the new model of the production planning system: the knowledge about the model should be shared in order to allow flexible and fast adaptations of the model with respect to the customer's needs.

Therefore, a specialized training was held at INFORM focussing on the following points:

- The basic theory of LPs resp. MIPs and their solution techniques
- Formulating small, well-defined requirements as LPs resp. MIPs (see [5])
- Modeling practical problems as LPs resp. MIPs
- Formulating LPs resp. MIPs with ZIMPL
- Solving LPs resp. MIPs with SCIP

This training was held in the form of 5 lectures (2 h each) accompanied by hands-on training, where participants could directly apply the topics discussed during the lectures. Here ZIMPL's easy-to-understand syntax was essential for the training's success. A more complicated modeling language could have easily discouraged many participants.

While this training cannot replace extensive practical experience when modeling linear programs and formulating them in ZIMPL, it allows a much broader discussion of modeling aspects with all people involved in the new optimization module of the production planning system. The main idea of the new module was to optimize the complete production plan simultaneously as opposed to previously known approaches optimizing in a more successive fashion.

After developing a small prototypic optimization module showing the potential improvement of a simultaneous approach in comparison to a classic successive approach, the decision was made to develop a full-scale optimization module

including many real-world restrictions (minimal production quantities, lead times, overtime, backlogging, ...).

The optimization module was developed by optimization experts in close collaboration with consultants, experts, and software developers of the production planning system, add*ONE. The training on modeling with ZIMPL held before was essential for this step, since it allowed all participants to understand the ZIMPL-models. Therefore, the discussion about the model's weaknesses or additional features was close to the particular model, shortening the time to rework and improve the model.

Testing a model naturally needs some data for the different parameters. Initially, standard test sets from the literature were used to evaluate different models. Furthermore, these were essential in comparing different solving heuristics, since the resulting MIPs were in general too large to be solved by a single run of a solver.

Additionally, data sets from the production planning system, add*ONE, with up to 2,000 products and 25 time periods were used to test the model and the solving heuristics. On the one hand, this allowed to fine-tune the solution heuristics to real-world problem sizes; on the other hand, the solutions for these data sets were closely inspected with experts from production planning and compared with solutions of the current optimization module to identify omissions and weaknesses in the model formulation. These led to an improved model, that was again tested on real-world data sets and inspected. This loop was repeated until a robust model was found, which led to solutions of largely improved quality compared to the current optimization module.

7.5 Conclusion

We have presented ZIMPL, a freely available AML with a very clear syntax close to the mathematical notation and powerful features. To demonstrate its usefulness we have shown step-by-step how a classical model of production planning, the MLCLSP, can be formulated im ZIMPL. This model was the basis of a new optimization module in the production planning system of INFORM GmbH.

The ease-of-use of ZIMPL together with its powerful features sped up the training and refinement process of the model substantially, as it allowed to rapidly change the model resp. the data. This very positive experience adds itself to the pool of successful applications of ZIMPL in real-world application. Therefore, we strongly recommend ZIMPL whenever an AML for a new optimization problem has to be chosen.

References

1. Achterberg, T.: Scip: Solving constraint integer programs. Mathematical Programming Computation **1**(1), 1–41 (2009)
2. Achterberg, T., Koch, T., Tuchscherer, A.: On the effects of minor changes in model formulations. Tech. Rep. 08-29, Zuse Institute Berlin, Takustr. 7, Berlin (2009); URL http://opus.kobv.de/zib/volltexte/2009/1115/

3. Berthold, T., Heinz, S., Vigerske, S.: Extending a CIP framework to solve MIQCPs. Tech. Rep. 09-23, ZIB, Takustr.7, 14195 Berlin (2009); URL http://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/1137
4. Bisschop, J., Meeraus, A.: On the development of a general algebraic modeling system in a strategic planning environment. Math. Program. Study **20**, 1–29 (1982)
5. Brown, G.G., Dell, R.F.: Formulating integer linear programs: A rogues' gallery. INFORMS Trans. Educ. **7**(2), 153–159 (2007)
6. Buschkühl, L., Sahling, F., Helber, S., Tempelmeier, H.: Dynamic capacitated lot-sizing problems: a classification and review of solution approaches. OR Spectrum **32**(2), 231–261 (2010)
7. Bussieck, M.R., Meeraus, A.: General algebraic modeling system (GAMS). In: Kallrath, J. (eds.) Modeling Languages in Mathematical Optimization, pp. 137–158. Kluwer (2004)
8. Cunningham, K., Schrage, L.: The LINGO algebraic modeling language. In: Kallrath, J. (eds.) Modeling Languages in Mathematical Optimization, pp. 159–171. Kluwer (2004)
9. Fourer, R., Gay, D.M., Kernighan, B.W.: A modeling language for mathematical programming. Manag. Sci. **36**(5), 519–554 (1990)
10. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming, 2nd edn. Brooks/Cole—Thomson Learning (2003)
11. Koch, T.: Rapid Mathematical Programming. Ph.D. thesis, Technische Universität Berlin (2004); URL http://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/834. ZIB-Report 04-58
12. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010. Math. Program. Comput. **3**, 103–163 (2011)
13. Pochet, Y., Wolsey, L.A.: Production Planning by Mixed Integer Programming. Springer, New York (2006)

Part III
Aspects of Modeling and Solving Real World Problems

In this third part we find discussions of various aspects of modeling and solving real world problems among them reasons for selecting a procedural programming language or a declarative modeling language (Chap. 8), data preparation (Chap. 9), visualization of data structures (Chap. 10), a feature wish list and reflections about the current and possible future role of algebraic modeling languages in Chaps. 11 and 12.

Chapter 8
Why Our Company Uses Programming Languages for Mathematical Modeling and Optimization

Hermann Gold

Abstract In this article the main reasons why Infineon Technologies uses programming languages for optimization in its semiconductor manufacturing facilities are presented. Experiences from the past with optimization techniques for logistical problems in our fabs are discussed and our choice of programming languages for optimization is explained as a strategic decision. A closer look at the scheduling and routing problems as they appear in our semiconductor fabs shows that this choice was also practically enforced, since we encountered certain specific and severe difficulties when we tried to work with a commercial Algebraic Modeling Language (AML) and the accompanying solver suite at one of our basic optimzation problems, namely a specific routing problem requiring load minimization, load homogeneity and fair queueing. The possible gain of new developments in AMLs for modeling transparancy in our company is disputed and we conclude that the incorporation of Brownian network formulations into AMLs could possibly relieve our optimization programmers' work.

8.1 Introduction

Semiconductor manufacturing would be too complex to be amenable to mathematical optimization was the prevailing opinion amongst managers and simulation experts in our company which had been accepted as a paradigm for many years. That is why that purely descriptive methods from queueing theory and discrete-event simulation were considered the most advanced help that mathematics could offer for the manufacturing management of our fabs. Meanwhile our opinion on what is amenable to mathematical optimization has changed fundamentally to a good deal due to impulses having come from Operations Research Society and in

H. Gold (✉)
Infineon Technologies AG, Wernerwerkstrasse 2, 93049 Regensburg, Germany
e-mail: hermann.gold@infineon.com

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_8, © Springer-Verlag Berlin Heidelberg 2012

particular out of the GOR working group "Real World Mathematical Optimization". We critically reflected the question whether semiconductor manufacturing should really be more complex, more stochastic, more nonlinear in behaviour than for instance the electricity business or the process industries. Of course the answer is "no", but why then does our company rely on programming languages when Algebraic Modeling Languages (AML) have been successfully introduced in those other industries?

8.2 An Answer Given by Experience from the Past

It was not the first time that our company tried to install a general optimization technique for the suite of logistical problems arising in its semiconductor fabs, when it started to build a proprietary solution to these problems based on Linear Programming in the year 2008. In those earlier trials different methods from Operations Research were applied, in particular: simulated annealing, discrete-event simulation and scheduling heuristics. Each of the implementations of these methods had its specific weaknesses.

Simulated annealing was much too time consuming to tackle the NP-hard problems considered, each in its full size. Therefore it had been required that a manufacturing expert, preferably the head of line control, should define what the key areas (i.e., the most critical areas to be taken into account) at the manufacturing floor would be. This sounded absurd to those experts because they were well aware of the fact that the choice of those key areas was a highly important and very sensible part of the problem. And so the project failed as far as optimization is concerned.

Discrete-event simulation was at least pretty good predictable for its computation times, but with its response times of about 7 h for a typical calculation instance related to a semiconductor fab producing power switching and logic devices it was still insatisfactory. Some decomposition ideas helped to reduce computation times by a factor of 2–3 for the category of stationary problems. For this problem category system states become time homogeneous and their probability distributions become independent of the initial states. This assumption is usually made out of the long and—in part—the medium term planner's perspective. Apart from decomposition ideas there were also vague hankerings for stronger parallelization even for essentially indivisible building blocks of a simulation scenario in order to cope with time complexity. But no concrete steps in that direction have been made and the simulation community appears to be not very optimistic about parallel discrete-event simulation (cf. the discussion in Fujimoto [5], where it is called into question whether the field will survive at all.). The major problem with discrete-event simulation is that its precision is very poor even from the perspective of a practitioner who doesn't care much about errors of say $\pm 0.5\%$ for machine utilizations, when the simulation run times are too small. This can be seen by just trying to reproduce the time dependant solution for the $M/M/1$-System (cf. Gross and Harris [9])—a very tiny system compared to an entire semiconductor fab

queueing model—via discrete event simulation. An ensemble of 10,000 runs is necessary to get an accuracy which fulfills the expectation of the practitioner. However the practice among experts who apply discrete-event simulation software tools for analysis is to work with ensemble averages based on just 10 runs or less, since results have to be produced in less than the duration of the worker's shift, e.g. 8 h, otherwise the results are useless. All in all, discrete-event simulation is a very robust and rigorous tool and one knows how it is done, but for the problem types considered in our logistics environment it is inappropriate.

In times where efficiency of equipment, use of consumables and energy have got most important for the operation of a fab, heuristical methods as a substitute for precise optimization are most questionable. Beyond that a great deal of discussions on the results for load distribution stemming from heuristics—and also on the ones stemming from simulation—has been around the questions "Why does it (the heursitic or simulation) produce this result? Why is the result so bad?". Of course this is unproductive in the long run, to say the least.

All three attempts made by our company to do planning and/or dispatching by means of an OR tool had in common that systems modeling was mainly driven by internal people and the analytical tool belonged to the realm of some external provider. Either party did not fully understand the intricacies of the counterpart's realm. A constellation like this can well be successful when the job is just to put some ready-made technique into practice with a proof of concept in the back. Since however a full proof of concept had not been available by then, the projects were involved with matters of investigation and development of new algorithms.

When the concept envisioned by a tool provider cannot fulfil the requirements of his customer in total there is need for concerted development under a common leadership and both partners must be prepared to adapt parts—maybe even essential parts—of their contribution for the sake of this concerted development. This means, for instance, under certain circumstances a vendor of a discrete-event simulation software would have to accept that the simulation engine needs to be enriched by components for graph theoretical methods to calculate connected components as a prerequisite to decomposition and hierarchical modeling. Assuming the willingness of an overseas simulation software provider company to open its code for external developers to enable them to program their own user-defined functions the possible range of application will broaden somewhat for the customer though it will still remain more or less in the field of computation which the original software developers could imagine.

Likewise the vendor of a Planning tool would have to accept that heuristical methods used to achieve load balancing objectives must be replaced by the appropriate exact solution technique when the gap between approximate solution and exact optimum appears to be significant to the customer. And, in anticipation of some matter of Sect. 8.3, a company providing optimization tools, might have to accept that a barrier method used to solve quadratic problems has to be replaced by a QP Solver using the Primal-Dual Interior Point Method when it is required by the type of problem considered. With some extraordinary engagement aspects of

the like should not represent a great hindrance to the introduction of a commercial optimization tool with its specific Algebraic Modeling Language (AML).

The great hindrance arises from the goal to improve the understanding of and the knowledge about the production process with the help of experiments. To accomplish this a general programming language without any restrictions is needed, so as to avoid any inhibitions in the flow of ideas.

8.3 Reasons Arising from Mathematical Modeling and Solver Aspects

The mathematical ground model we use for the optimization of our fabs has been explained in Gold [6] and Romauch et al. [16]. The queueing theoretic basics for the dimensioning of buffers have been explained in Gold and Frötschl [7] and Gold [8]. The accomanying method for scheduling has been patented under the European Patent $EP\,1\,567\,921\,B1$. To make this article self-contained the main ideas are repeated where necessary. In Gold [6] it has been described how a primal dual interior point optimization method is applied to solve a quadratic optimization problem with a quadratic objective function designed to distribute load in a closed set of machines such that load is minimized and homogenized. More precisely, homogeneity of load should be realized at the lowest possible utilization levels. The constraints of the optimization problem are linear and hence the Hessian matrix has to be calculated only once for a given instance. Furthermore the special structure of the objective function can be exploited to calculate the Hessian most efficiently. When I first programmed the Hessian matrix calculation in year 2002 using a general algorithm from a text book, instances with just about 600 variables needed 30 min of computation time for optimization. The programming code, which I further specialized for the objective function given, had been practically useful already for instances up to 1,500 variables. Still it was clear that this computational speed would be insufficient for broad use in practice, it could rather be used for special questions and some types of ad-hoc analyzes. When we contacted a commercial vendor of optimization tools it turned out that he was struggling with the same running time complexity problem. Run time grew exponentially with the number of variables and system size was limited by a maximum number of 3,600 variables. The chart showing running time as a function of the number of variables, the number of constraints being equal to the square root of the number of variables, is given in Fig. 8.1.

The instances created are rather trivial: All machines of a closed machine set are fully flexible, i.e., there is no dedicated traffic. Of course this kwowledge is not incorporated in the programming code.

Fortunately we could reformulate the objective in a way which keeps up the homogeneity goal while getting rid of the quadratic terms in the objective function. The reformulation we arrived at was as follows:

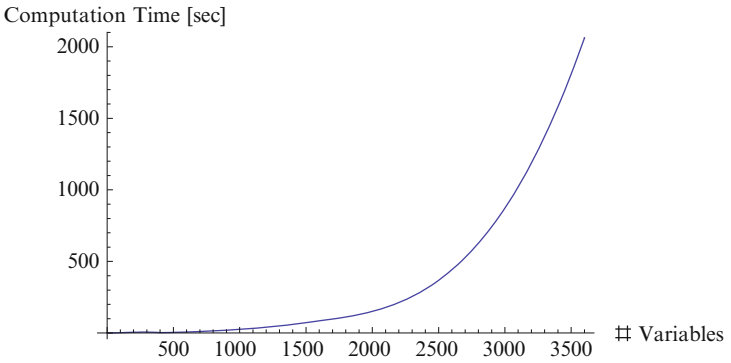


Fig. 8.1 Computation times for regularly structured test problems as a function of the number of nonzero variables for the QP solution using the barrier, primal and dual simplex method

Let the gap for a machine be the difference between its availability and its utilization. Now apply the following rule:

"In a closed set of machines with n relevant machines, first maximize the gap of the group of machines which are worst-off (in the sense that a policy achieving load minimization and load homogeneity would load this group of machines most heavily); second for fair queueing conditions of the job classes attending the worst-off machine group, maximize the gap of the second worst-off machine group, and so on until the last case which is, for fair queueing conditions of the job classes attending all of the preceding $n-1$ machine groups maximize the gap of the best-off machine group."

This idea was adapted from the theory of justice by John Rawls. Substituting the term "closed machine set" by "basic structure", "machine" by "representative", "gap" by "welfare" and "fair queueing" by "equal welfare" the quoted sentence can be read in Rawls [15], Chap. 2, Sect. 13. The first queueing theorists who applied this fairness principle as a design objective for data communication networks are Bertsekas and Gallager [2].

The runtime improvement we have got through this reformulation is immense. The chart showing running time as a function of the number of variables is given in Fig. 8.2. The instances had been created by means of a random number generator which was guided in a way so that the CMS parameters created are realistic for semiconductor fabs. This means that each job class has its specific dedication pattern and process speeds depend both on machines and job classes. Furthermore the average number of machines released for some job class is rather small ($\approx$3 in our test cases) so that the matrices related to the LP-Problems are sparse. The gain in computation speed is of course also realized for the instances related to Fig. 8.1, in fact it is even slightly higher.

Could we have made profit from this insight when we had worked with the commercial vendor's solver suite? Unfortunately not, since we would have had to make sacrifices to the fair queueing condition of the job classes when we had

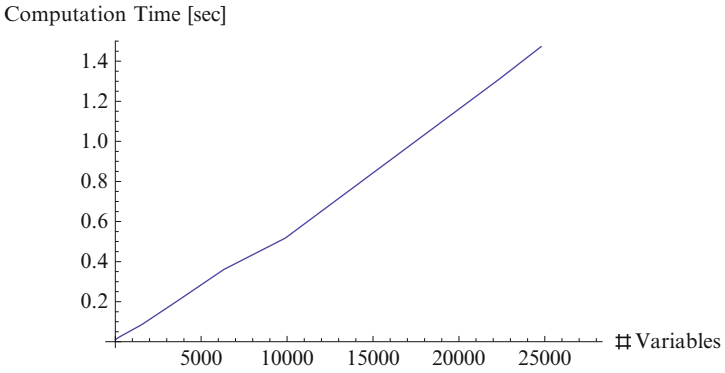


Fig. 8.2 Computation times for problems arising in semiconductor front end manufacturing as a function of the number of nonzero variables for the LP solution using primal-dual interior point methods

switched to use his LP solver, with its solutions looking pretty much like Simplex solutions rather than Interior Point solutions.

Why is the choice of the primal dual interior point method so important to the type of queueing problems we are tackled with in our fabs? One important aspect of our fabs is that jobs change job classes while travelling through the production network. At each stage of the production process a manufacturing lot belongs to a certain job class which can be processed on an arbitrarily fixed subset of the total set of machines. The best answer to the stochasticity of arrivals as they occur in these queueing networks with their reentrant flow is to maximize the entropy in the routing decision at the level of a nominal plan. This maximization of entropy is also implied by the load homogeneity condition and fortunately it can be achieved by exploring into the analytical center of a solution facet when there are multiple solutions to a minimization problem without formulating the objective explicitly for the objective function of the LP optimization problem. The result is an activity vector with associated routing matrix which guarantees the largest degree of resource pooling (according to the concepts presented in Kelly and Laws [13]) for each given closed machine set. Note that the concept of resource pooling also applies to routing alternatives which stretch over several processing steps. Its importance even grows in these settings since they become sensitive to second-order statical data, beyond mean value parameters (see Dai et al. [4]).

The programming tools we work with are MatLab for productive operation and additionally Mathematica for development. Their built-in Interior Point LP Solvers are very similar to the IBM Freeware CLP Solver as far as computation time and accuracy of solution are concerned. Although both tools are missing a uniform approach to optimization solution processes, they can be considered as strong modeling languages (see Kallrath [11], Chap. 19, for a discussion of MatLab as a modeling language and its use in the TOMLAB optimization environment).

Let me now turn to the aspect of writing code for an optimization problem. Practically seen there was no big difference for me in writing the optimization problem given at the lowest level of the decomposition hierarchy, namely at the level of closed machine sets, in a programming tool like MatLab or Mathematica or with the Algebraic Modeling Language which I had the opportunity to work with. For an elementary module I reimplemented my programming code for Mathematica in a commercial tool with AML with the help of an expert. After all, from the standpoint of a program developer I saw only minor advantages, e.g., in the fields of indexing data, which would draw me to learn to work with the specific AML. The advantages of having the full flexibility coming with a programming language and to be able to resort to built-in methods, e.g., from combinatorics, from graph theory or from theory of transforms outweighs this attraction.

8.4 The Aspect of Modeling Transparancy

The company's point of view should however be somewhat different. As I indicated in the beginning of this article I think there is no unifying theory on the physics of a factory. From a strict point of view the contribution of a mathematical modeller to the improvement of factory operation lies in the solution of subproblems in a way such that medium and long-term performance characteristics are optimized. There is no global optimum to the scheduling problem as it occurs in semiconductor manufacturing (cf. Chen and Yao [3], Chap. 12). In the case of our proprietary logistic solution, mainly knowledge and methods from queueing theory, optimization theory and graph theory are combined together to keep a sample path on the track towards the long-term optimum. If so, there is a need to have checks and balances. And the best basis for this in turn would be to have a common modeling language which is also understood by people from manufacturing floor and managers apart from the programmers themselves, as suggested by Beisiegel [1]. However our practical experience did not match this ideal. In the case of the load minimization and homogenization problem we made the following experience. You can certainly explain the concept of closed machine sets to managers and operating staff by drawing examples of the connected components of a fab graph they are related to, but you will have to do this in an intuitive way and will have to find the right vocabulary and pictures. In a similar vein it would be very difficult to explain the way we calculate the dynamics of the whole fab based on Brownian Network Approximation, as described in Harrison [10]. What users want to know is the mapping between the scheduling policies in the modeling world and the real world of our fabs. For instance with regard to the non-idling condition, i.e. no machine should idle when there is work available to be processed on it, they are not willing to read it out of the mathematical formulation in the Brownian Network, which states $\int Q(t)dY(t) = 0$, where $Q(t)$ is the queue length process and $Y(t)$ is the idleness process. They do want to know how much idealism is comprised in

our mathematical formulation of the non-idling condition and where precesely it is present by an explanation in terms they are familiar with.

We conclude that an algebraic modeling language is far too formal for the purpose of checking with managers and users whether the mathematical model meets their requirements. An intermediate language close to how the solution of a problem would be explained to managers together with a translation mechanism into an AML, as described by Neumeier [14] could be a step towards modeling transparancy.

The place where AML's could help our company a lot is among the programmers themselves dealing with optimization. A proper extension of an AML which comfortably allows to formulate Brownian Network Approximations of stochastic processing networks could prepare the way of an AML into our company.

In a broader view it would mean to follow the concept of polylithic modeling as described in Kallrath [12]. The application of corresponding solution appraoches in the energy industry and in a water reservoir problem reported there gives rise to the hope, that an approach more exact than the one given in Harrison [10] to the problem of translating the optimal policy for the fluid model, which corresponds to the above mentioned Brownian network approximation, back to the original problem of interest, might be possible.

References

1. Beisiegel, B.: Modeling Languages: What I liked in the Past and what I hope for the Future, 85th Meeting of the GOR Working Real World Optimization, Physikzentrum Bad Honnef (2010)
2. Bertsekas, D., Gallager, R.: Data Networks. Prentice Hall, Englewood Cliffs, NJ (1987)
3. Chen, H., Yao, D.D.: Fundamentals of Queueing Networks. Springer, New York (2001)
4. Dai, J.G., Hasenbein, J.J., Kim, B.: Stability of join-the-shortest-queue-networks. Queueing syst. **57**, 129–145 (2007)
5. Fujimoto, R.M.: Parallel discrete event simulation: will the field survive? ORSA J. Comput. **5**(3), 213–230 (1993)
6. Gold, H.: Dynamic Optimization of Routing in a Semiconductor Manufacturing Plant. Operations Research Proceedings 2004. Springer, Heidelberg (2004)
7. Gold, H., Frötschl, B.: Performance Analysis of a Batch Service System. In: Labetoulle, J., Roberts, J.W. (eds.), The Fundamental Role of Teletraffic in the Evolution of Telecommunications Networks. Vol. I, pp. 155–168. Elsevier, Amsterdam (1994)
8. Gold, H.: A markovian single server with upstream job and downstream demand arrival stream. Queueing Syst. **30**(4), 435–455 (1998)
9. Gross, D., Harris, C.M.: Fundamentals of Queueing Theory. Wiley, New York (1988)
10. Harrison, J.M.: The BIGSTEP approach to flow management in stochastic processing networks. In: Kelly, F., Zachary, S., Ziendins, I. (eds.), Stochastic Networks: Theory and Applications Oxford University Press, Oxford, 57–90 (1996)
11. Kallrath, J. (ed.): Modeling Languages in Mathematical Optimization. Kluwer Academic Publishers, Norwell, MA, USA (2004)
12. Kallrath, J.: Polylithic Modeling and Solution Approaches Using Algebraic Modeling Systems. Optimization Lett. **5**(3), 453–466 (2011)

13. Kelly, F.P., Laws, C.N.: Dynamic routing in open queueing networks: Brownian models, cut constraints and resource pooling. Queueing Syst. **13**(4), 47–86 (1993)
14. Neumaier, A.: FMathL—Formal Mathematical Language, 85th Meeting of the GOR Working Group Real World Optimization, Physikzentrum Bad Honnef (2010)
15. Rawls, J.: A Theory of Justice, Belknap Press (1971)
16. Romauch, M., Gold, H., Laure, W., Seidel, G.: Advanced Equipment Capacity Planning based on Mathematical Modeling. Semiconductor Conference Dresden (2009)

Chapter 9
Data Preparation for Optimization with R

Hans-Joachim Pitz

Abstract This chapter deals with GNU R as a tool for data preparation in optimization modeling. For statistical applications R is a widely used tool including a rich variety of different techniques and procedures. Due to the excellent features for data handling and manipulation R is highly recommendable also for other fields of application with extensive requirements in data processing. The preparation of optimization data in an industrial environment is one of these fields.

9.1 Introduction

In practice, data preparation for optimization applications frequently causes much more work than expected. In the majority of cases, the data has to be read from different sources. Unfortunately their contents is often not consistent. Reasons for example can be different names or numbers for the same products, missing, incomplete or wrong part lists or recipes, out-dated data or data that was determined with insufficient quality etc. So it might be a long way till an optimization can be started. After collecting the data, the errors have to be identified and corrected if possible, data sometimes has to be supplemented by calculations, data has to be assembled to the data model that has finally to be transfered to the solver directly or via a modeling language. Particularly if the data model also has to describe the structure of the optimization model, like for state task models, additional data manipulations are required. However to gain acceptance for optimization applications it is important that all data manipulations are done without manual interaction. Since `R` is assigned to statistics, it is not obvious that it is an outstanding tool for these tasks. The intention of this article is to introduce some of the `R` features for preparing optimization data in a short overview without going into specifics.

H.-J. Pitz (✉)
BASF SE, KTE, 67056 Ludwigshafen, Germany
e-mail: hans-joachim.pitz@basf.com

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_9, © Springer-Verlag Berlin Heidelberg 2012

A great deal of literature is available providing details, to which reference is made on the R home page [8].

9.2 What is R?

R is an Open Source software [8], which is mostly categorized as statistical software. However, this only describes part of the features R provides. It is also a highly flexible programming language for a wide range of data analyses, data manipulations and generation of sophisticated graphics. Flexible data communication ensures that many different data sources can be accessed. The possibility of incorporating packages opens up access to state-of-the art methods and algorithms [2, 4] from a multitude of application areas that extend far beyond statistics. This provides the best prerequisite for efficient data preparation. Although R was developped as a tool to support teaching, the number of commercial and industrial applications increases steadily.

9.3 The History of R

R is a free implementation that essentially corresponds with the published descriptions of the S language. The S programming language was developed by Bell Laboratories at AT&T from 1976 onwards. Rick Becker and John Chambers were the "main implementers" [1] at Bell Labs. Since 2008 Tibco Software Inc. has had the rights to the original S code, and markets the program as commercial software under the name "Tibco Spotfire S+". It is worth noting that John Chambers received the "1998 ACM SOFTWARE SYSTEM AWARD" of the Association for Computing Machinery (ACM) for the development of S. In the reason it was stated: "Dr. Chambers' work will forever alter the way people analyze, visualize, and manipulate data. . . S is an elegant, widely accepted, and enduring software system, with conceptual integrity," [7]. S was therefore the first statistical software to receive the top software award from the ACM.

The R language was developed from 1992 onwards on the basis of the ideas and syntax of the S language. This development was originally carried out for teaching purposes by Ross Ihaka and Robert Gentleman in Auckland. Important milestones are the publication of R under GPL 1995, and the establishment of the "R Development Core Team" in 1997 in which John Chambers is also involved. In 1997 the Comprehensive R Archive Network (CRAN) was established, a network of servers for distributing the software and additional packages.

The majority of statistical standard procedures are contained within the basic R installation. Special applications and procedures that also cover other work areas are made available via CRAN. At the beginning of 2011 there were more than 2,900 packages on CRAN and the number is growing exponentially. There are other

repositories dedicated to special applications such as Bioconductor, in which further packages are provided. The feature for the simple creation and distribution of new packages via CRAN is a success factor that should not be underestimated. It is certainly one of the reasons for the fast speed with which R has spread.

9.4 How can the Preparation of Optimization Data be Supported with R?

In the industrial use of optimization, providing correct and consistent data is frequently more time-consuming than creating the mathematical model. Master data for describing products and systems (e.g. part lists, recipes, throughputs and capacities) is needed as well as dynamic data (e.g. order data or measuring data). The sources of this data are: Enterprise Resource Planning Systems (ERP) such as SAP in which the business processes are depicted, Manufacturing Execution Systems (MES) and Plant Information Management Systems (PIMS) in which the production process data is collected. Unfortunately, this data very often contains inconsistent, incomplete or wrong information. Before creating optimization input data, an intensive data analysis must therefore be carried out. In the first step this has to be done interactively till all error sources have been detected. After all obstacles are known this process can be automated. Even though the use of spreadsheets is extremely popular in industrial practice, they are hardly adequate for efficient preparation of optimization data. The data preparation requirements are met in a considerably better way by using databases. R, on the other hand, combines the capabilities of spreadsheet and database programs with many other applications that are or could be needed for preparing optimization data [3]. The following list describes the procedure of a data preparation for optimization:

1. Read in the data from databases and spreadsheets.
2. Completeness and consistency check for all data, automatic data correction if possible, error reports.
3. Determination of additional information that is implicitly contained within the data but is explicitly required. A simple example of this is determining the membership of products to product groups from parts lists in order to support campaign production.
4. Additional calculations that are not within the scope of the optimization but are necessary prerequisites for producing reliable optimization data. For production planning this can be a minimum inventory level, determined from historical forecast errors or automated forecasts. For production optimization measuring data has to be error corrected and for continuous productions being split up into steady-state intervals.
5. Assembly of data model.
6. Output of the data model to serve modeling language or solver.
7. After optimization a post-processing of the results has to be done for example to visualize the results.

The above mentioned tasks have been chosen to explain the comprehensive features of R.

- Communication with databases and spreadsheets (items 1,7)
  R provides the communication features with databases via ODBC and JDBC drivers, or via direct interfaces to Oracle, MySQL, PostgreSQL, H2, SQLite. Data can also be exported to and imported from Excel. XML or compressed files can also be read and written [6]. These capabilities are extremely useful for assembling the input data for optimization, since the data normally has to be gathered from different data sources and therefore intermediate steps for reformatting the data are not required. However, these communication capabilities are also important after postprocessing the results in R when data has to be transferred to further systems for using the results. With increasing data connection requirements the available functions for reading and writing binary data can be used in order to provide interfaces to proprietary binary files [6].
- Variable data manipulation (items 2,3,4,5)
  The indexing of tables and multi-dimensional arrays is one of the features of R which you really learn to appreciate when you are processing large numbers of linked tables. Rows and columns can be accessed via their number, name or via Boolean vectors. This makes the manipulation of large quantities of data extremely variable and easy, even thought R is not a relational data base program. Appropriate functions for selecting subsets, for aggregating with any functions (including self-defined ones) and for reshaping tables are also available. Comprehensive data manipulations are always needed if the data has to be enriched with information from the optimization model, as needed for state task models (Fig. 9.1) in which the system structures are depicted by "parts lists". The R code for these kind of data models is easy to produce and simple to maintain.

  If table sizes exceed the capacity of the RAM, the tables can easily be stored in a database so that the actually required quantities can be selected using SQL queries.
- String processing (items 2, 3, 5, 6)
  R provides effective tools for analyzing and processing character strings. They are needed to analyze and standardize names, to determine information from product names and recipes / part lists, to generate state and task names and to write data models using the syntax rules of modeling languages. Functions for the data model output using the GAMS syntax is shown as an example in the appendix.
- Graphics (items 4, 7)
  A picture says more than a thousand words, and speeds up the evaluation of results if they consist of large quantities of data. Graphical analyses can be used both to monitor data preparation and also to visualize the results.

  In Fig. 9.2 results are displayed that were aggregated by product groups to display the monthly production quantities for the product groups. This allows checking the campaign sequences.

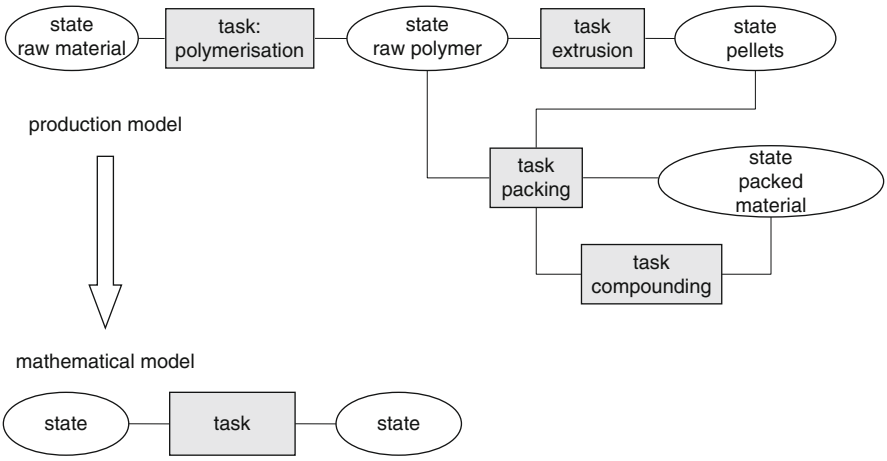


Fig. 9.1 State task network

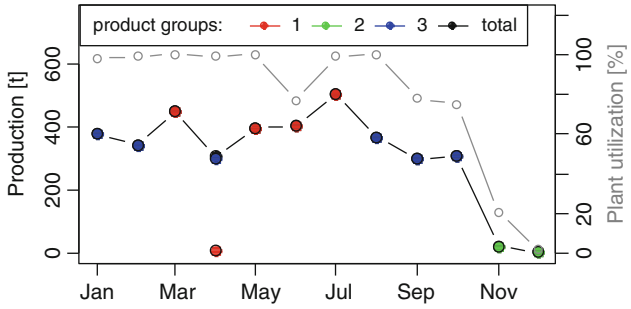


Fig. 9.2 Campaign planning

There are many ways of generating high-quality presentation graphics. It is also worth mentioning the ease with which graphical mass evaluations can be produced, in which pdf documents with any number of graphics can be generated. Graphics can also be automatically documented together with tables and numerical results in standard reports. The required report templates have to be provided in Latex so that they can then be automatically supplemented with the latest calculation results. This is done using Sweave [5], which is part of the standard installation of R. The same also applies to texts in Open Document Format (ODF), for which the odfWweave package is then required instead of Sweave.

- Methods of analyzing data (item 4)
  The data requires more or less preparation depending on the application and the data sources. R provides a multitude of procedures for an extremely broad range of applications, meaning that the wheel doesn't have to be repeatedly reinvented

but just used. However, we can assume that practically all fundamental methods for data analysis are available. Some analyses carried out within the scope of preparing optimization data will be shown:

- Sales forecasts are needed to plan *Make-to-Stock* products. If the forecasts (Fig. 9.3) are to be produced for a large number of products, the use of powerful statistical procedures is of considerable help in determining a reliable data basis for the optimization.
- Profit contributions are a necessary input parameter for profit-oriented planning. Frequently there is not a single value for profit contribution but the data is statistically distributed, due to the different selling prices and logistical costs. For tactical planning customer orders are not yet available. In order to be able to optimize the production portfolio the profit contribution distributions (Fig. 9.4) must be determined and taken into consideration.
- In the optimization of continuous production processes, measurements have to be assigned to quasi-stationary time ranges (Fig. 9.5). These different ranges are caused by modifying production parameters such as the throughput. A manual assignment appears to be quite simple, but an automation requires sophisticated algorithms [13].
- Measurements from production processes can be monitored and corrected using physical equations such as masses and energy balances. The procedure used is known as *Data Reconciliation*.

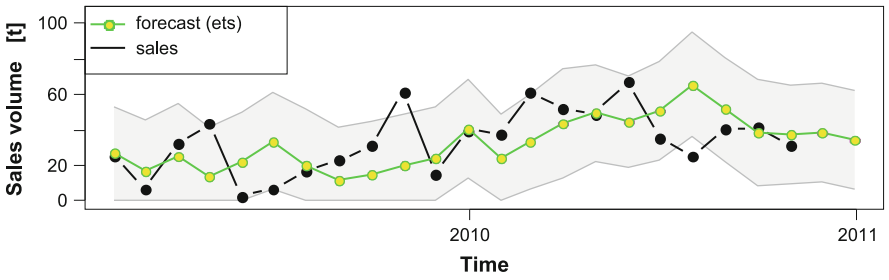


Fig. 9.3 Univariant forecast

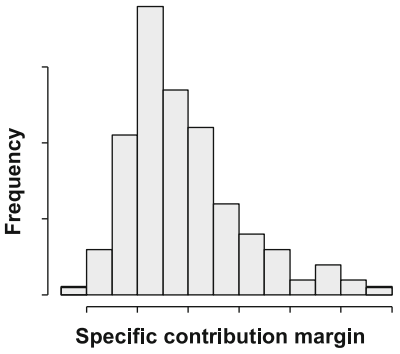


Fig. 9.4 Distribution of the contribution margins of a product

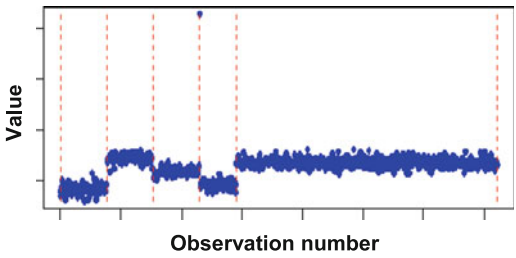


Fig. 9.5 Steady-state time ranges

9.5 How can R be Linked to Optimization Software

Besides data preparation R also provides optimization procedures and links to external solvers. Because of the large number of R packages, it is not always easy to find the appropriate packages for a topic. To overcome this problem CRAN offers "Task Views" that contain summaries of the available packages for different work areas, such as optimization. Subsequently some available procedures for processing the prepared optimization input are introduced.

1. If the objective function vector and constraint matrix were assembled in R, these could be directly solved. Obviously this is only reasonable for those models for which modeling languages do not provide a real advantage. For those applications two procedures are provided:
   a. Call of solvers via direct interfaces (currently available: CPLEX, GLPK, Symphony, lpSolve).
   b. Output of data in MPS format for further processing.
2. When using GLPK [10,11] as modeling language and solver all required functions will be provided by the GLPK package. The syntax of the GLPK language corresponds to a subset of AMPL and is only applicable to linear problems. The steps of such a process are:
   a. Read in the GLPK model using the GLPK package.
   b. Create the data model with the GLPK package.
   c. Solve the LP within R using the GLPK solver. However, the solver is only suitable for small to medium-sized MILP problems, since the computing time increases disproportionately with the number of binary variables in comparison to CPLEX.
   An alternative is:
   Write the model in MPS format (fixed or free) or CPLEX LP format using the GLPK package, in order to apply a separate solver.
3. Write the data (sets and parameters) in the format of an algebraic programming language. This method is versatile, since data formats for any algebraic programming language can be simply provided. However, there are currently no packages on CRAN available for writing data using the syntax of modeling languages.

The appendix contains the documentation of example functions covering that task for GAMS input formats. These functions also give a small insight into the processing of tables and strings with R.

4. Another option is to write the data in databases, spreadsheets or comma-separated files (csv) files. These data sources can be read by modeling languages. In principle, this is very similar to the previous point. The difference is the data does not follow the syntax rules of the modeling languages.
5. The most advanced link between R and optimization software is provided by GAMS [12]. Their R package "gdxrrw" enables reading and writing of binary GAMS data exchange (GDX) files for creating input data as well as for reading optimization results.

9.6 Summary

R is a very efficient tool for preparing optimization data. If data does not have to be pre- or post-processed using statistical methods R is one tool among others. In this case it is mainly a matter of taste and routine which one is preferred. If statistical analyses are additionally required R seems to be the most powerful open source software available for this kind of applications. To illustrate statistical analyses that may be needed prior to optimization some examples have been introduced. For production planning forecasting and the determination of safety stocks were mentioned. For the optimization of continuous production processes the determination of steady state ranges as well as data reconciliation have been listed.

Appendix A: R Application for Generating a GAMS Data File

This following example R code generates data that has to be integrated in the GAMS model by using an include statement. Here the basic figures directly have been given in that code. In a real world application this is the point where the data preparation has to be done.

```
### Data for the transportation problem (adapted from Dantzig, 1963)
### taken from GAMS tutorial
#
dfPlant     <-   data.frame(i=c("seattle", "san-diego"),
                            a=c(350,600), stringsAsFactors=FALSE)
dfMarket    <-   data.frame(j=c("new-york", "chicago", "topeka"),
                            b=c(325,300,275), stringsAsFactors=FALSE)
dfDist      <-   data.frame(i=rep(c("seattle","san-diego"), times=c(3,3)),
                             j=rep(c("new-york", "chicago", "topeka"), times=2),
                             d=c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4),
                             stringsAsFactors=FALSE)
#
### Write sets
```

```
#
WriteSet_GAMS(DatOut, SetNam="i", Set=dfPlant[,"i"])
WriteSet_GAMS(DatOut, SetNam="j", Set=dfMarket[,"j"])
#
### Write parameters
#
WriteParam_GAMS(DatOut, ParaNam="a", IndNam="i", Ind=dfPlant[,"i"],
               Value=dfPlant[,"a"])
WriteParam_GAMS(DatOut, ParaNam="b", IndNam="j", Ind=dfMarket[,"j"],
                Value=dfMarket[,"b"])
WriteParam_GAMS(DatOut, ParaNam="d", IndNam=c("i","j"),
                Ind=dfDist[,c("i","j")], Value=dfDist$d)
#
### Write scalars
#
WriteParam_GAMS(DatOut, ParaNam="f", Value="1000")
```

Appendix B: R-Output of the GAMS Include File

This GAMS data input is the result of the R code shown in the first appendix.

```
set i         / seattle
                san-diego /;
set j         / new-york
                chicago
                topeka   /;
parameter a     (i)
              / (seattle  ) = 350
                (san-diego) = 600 /;
parameter a     (j)
              / (new-york) = 325
                (chicago ) = 300
                (topeka  ) = 275 /;
parameter d     (i . j)
              / (seattle   . new-york) = 2.5
                (seattle   . chicago ) = 1.7
                (seattle   . topeka  ) = 1.8
                (san-diego . new-york) = 3.5
                (san-diego . chicago ) = 1.8
                (san-diego . topeka  ) = 1.4 /;
parameter f   / 1000 /;
```

Appendix C: R Functions for Writing GAMS Sets and Parameters

These two functions were used to write the data meeting the GAMS syntax rules. However, it is simple to adapt this code to the syntax rules of other modeling languages.

```
#--------1---------2---------3---------4---------5---------6---------7---------8

WriteSet_GAMS   <-   function(DatOut, SetNam, IndNam, Set, ncData=25,
                              bEmpty=TRUE){
```

```
#
### Write sets in the GAMS format
#   ============================
#
#   Input
#   DatOut    -   Name of the file, to which data should be appended
#   SetNam    -   Set name that should be used for the output
#                 In GAMS the set name must not exceed 63 characters.
#   IndNam    -   Vector with index names(Setnamen): c("i", "j", ...)
#                 for multi-dimensional sets. If the index names are the
#                 same as the column names of the data.frame Set
#                 this field can be omitted.
#   Set       -   one dimensional sets  : vector with set elements
#                 multi-dimensional sets: data.frame with columns of set
#                                         elements
#                 In GAMS the set elements name must not exceed 63 characters.
#   ncData    -   First column with set elements
#   bEmpty    -   Boolean variable to indicate whether an parameter statement
#                 without values should be printed
#                 bEmpty = TRUE    empty parameter statement will be printed
#                 bEmpty = FALSE   empty parameter statements will be omitted
#                 If bEmpty is TRUE then in GAMS the option "$onempty"
#                 has to be used
#
#   Output
#   The input text for the GAMS file will be returned. If input variable
#   "DatOut" is not missing the set information will additionally be
#   appended to this file.
#
### Variable Set will be coerced to class data.frame
#
Set        <-   unique(Set)
dfSet      <-   data.frame(Set)
nInd       <-   ncol(dfSet)
nSet       <-   nrow(dfSet)
#
### Without the option "$onempty" GAMS does not accept empty declarations
### For bEmpty=FALSE empty sets have to be omitted
#
if(nrow(dfSet)==0 & !bEmpty) return(invisible(NULL))
#
### Coercing elements of multi-dimensional set to character variables
### of the same length
#
if(nSet > 0) {
   mSet    <-   apply(X=dfSet, MARGIN=2,
                      FUN=function(x) format(x, digits=max(nchar(x))))
   dfSet   <-   as.data.frame(matrix(mSet, ncol=nInd), stringsAsFactors=FALSE)
}
#
### Collapse multi-dimensional sets to one character variable
#
cSet       <-   apply(X=dfSet,MARGIN=1,FUN=function(x) paste(x,collapse=" . "))
#
### Nonemty multi-dimensional sets wil be bracketed
#
if(nInd > 1 & nSet > 0)
   cSet    <-   sapply(X=1:nSet,
                       FUN=function(i) paste("(", cSet[i], ")", sep=""))
#
### Multi-dimensional sets need to get a heading line with index names
#
if(nInd > 1){
   if(missing(IndNam)) IndNam <- colnames(Set)
   cSet    <-   c(paste("(", paste(IndNam, collapse=" , "), ")", sep=""),
                  cSet)
}
#
### Preceding spaces of length "set 'SetNam' := "
```

```
#
if((nchar(SetNam) + 8) > ncData)
   ncData  <-   nchar(SetNam) + 8
cSet       <-   paste(paste(rep(" ",times=ncData),collapse=""), cSet, sep="")
#
### An blank line ist added to empty multi-dimensional sets
#
if(nInd > 1 & nSet == 0)
   cSet[2] <-   paste(rep(" ", times=ncData), collapse="")
#
### Add set statement in the first line "set 'SetNam'"
#
substring(cSet[1], 1, nchar(SetNam)+4)     <-   paste("set", SetNam, sep=" ")
if(nInd==1)
   substring(cSet[1], ncData-1, ncData-1)  <-   "/"  else
   substring(cSet[2], ncData-1, ncData-1)  <-   "/"
#
### The last line has to end with "\;"
#
nS         <-   length(cSet)
cSet[nS]   <-   paste(cSet[length(cSet)], " /;", sep="")
#
### Append set to the file "DatOut"
#
if(!missing(DatOut)) write(cSet, file=DatOut, append=TRUE)
#
### end of function
#
invisible(cSet)}

#--------1---------2---------3---------4---------5---------6---------7---------8

WriteParam_GAMS   <-   function(DatOut, ParaNam, IndNam, Ind, Value,
                           ncData=25, bEmpty=TRUE){
#
### Write parameters in the GAMS format
#   ===================================
#
#   Input
#   DatOut    -   Name of the file, to which data should be appended
#   ParaNam   -   Name of parameters, must not
#                 In GAMS these names are limited to 63 characters.
#   Ind       -   Vector or data.frame with sets used as indices
#                 If a scalar should be printed Ind has to be omitted.
#   IndNam    -   Vector with mit index names (set names): c("i", "j", ...)
#                 If the indexnames are the same as the column names of
#                 the data.frame Ind this field can be omitted.
#   Value     -   Vector containing the parameter values
#   ncData    -   First column with input data
#                 When the keywords start in the first column and the
#                 associated data always in the same column behind the keywords
#                 the data file will more readable.
#   bEmpty    -   Boolean variable to indicate whether an parameter statement
#                 without values should be printed
#                 bEmpty = TRUE    Empty parameter statement will be printed
#                 bEmpty = FALSE   Empty parameter statements will be omitted
#                 If bEmpty is TRUE then in GAMS the option "$onempty"
#                 has to be used
#
#   Output
#   The input text for the GAMS file will be returned. If input variable
#   "DatOut" is not missing the set information will additionally be
#   appended to this file.
#
#
### Without the option "$onempty" GAMS does not accept empty declarations
### For bEmpty=FALSE empty sets have to be omitted
#
if(!missing(Ind)) {
```

```
   #
   ### Empty multi-dimensional parameters
   #
   Ind                 <-   unique(Ind)
   dfParam             <-   data.frame(Ind)
   if(is.null(Ind) | (nrow(dfParam)==0 & !bEmpty)) return(invisible(NULL))
} else
   #
   ### Empty scalars
   #
   if(length(Value) == 0)                          return(invisible(NULL))
#
### Initializing of the index names for parameters and of the
### keyword for the parameter statement
#
if(missing(IndNam) & !missing(Ind))
   IndNam              <-   colnames(Ind)
cParaNam               <-   paste("parameter", ParaNam, sep=" ")
#
### Set dependant parameter data: indices (set names) and values
#
if(!missing(Ind)) {
   if(nrow(dfParam) > 0) {
      mParam           <-   apply(X=dfParam, MARGIN=2,
                                  FUN=function(x) format(x,digits=max(nchar(x))))
      dfParam          <-   as.data.frame(matrix(mParam, ncol=length(IndNam)),
                                             stringsAsFactors=FALSE)
      cParam           <-   apply(X=dfParam, MARGIN=1,
                                  FUN=function(x) paste(x, collapse=" . "))
      cParam           <-   c(paste(IndNam, collapse=" , "), cParam)
   } else
      cParam           <-   c(paste(IndNam, collapse=" , "))
   cParam              <-   paste("(", cParam, ")", sep="")
   if(length(Value) > 0) {
      iInd             <-   2:length(cParam)
      if(is.data.frame(Value))
         Value         <-   as.vector(Value[,1])
      cParam[iInd]     <-   paste(cParam[iInd], " = ", Value, sep="")
   } else
      cParam[2]        <-   " "
#
### Set independant parameter data: Scalars
#
} else
   cParam              <-   as.character(Value)
#
#
### Preceding spaces of length "parameter 'ParamNam'   /"
#
if((nchar(cParaNam) + 2) > ncData)
   ncData              <-  nchar(cParaNam) + 2
cParam                 <-   paste(paste(rep(" ",times=ncData),collapse=""),
                                  cParam, sep="")
#
### Adding the set statement "param 'cParaNam'"
#
substring(cParam[1], 1, nchar(cParaNam))  <-   cParaNam
#
if(length(cParam)==1 & !missing(Ind))
   cParam[2]           <-   paste(rep(" ",times=ncData),collapse="")
if(!missing(Ind))
   i                   <-   2   else
   i                   <-   1
substring(cParam[i], ncData-1, ncData-1)  <-   "/"
#
### The last line has to end with "\;"
#
cParam[length(cParam)] <-   paste(cParam[length(cParam)], " /;", sep="")
#
```

```
### Append set to the file "DatOut"
#
if(!missing(DatOut))
   write(cParam, file=DatOut, append=TRUE)
#
### end of function
#
invisible(cParam)}

#--------1---------2---------3---------4---------5---------6---------7---------8
```

References

1. Chambers, J.: Software for Data Analysis, Programming with R. Springer, New York (2008)
2. Venables, W.N., Ripley, B.D.: Modern Applied Statistics with S. 4th Edition, Springer, New York (2002)
3. Phil Spector: Data Manipulation with R. Springer, Carey, NC (2008)
4. Adler, J.: R in a Nutshell. O'Reilly, Sebastopol, CA, (2010)
5. Leisch, F.: Sweave: Dynamic generation of statistical reports using literate data analysis. In: Wolfgang Hŕdle and Bernd Rnz, editors, Compstat 2002—Proceedings in Computational Statistics, 575–580. Physica, Heidelberg (2002)
6. R Data Import and Export Manual. http://cran.r-project.org/
7. ACM Press Release, March 23, 1999, New York http://www.acm.org/announcements/ss99.html
8. Informational R Web Site. http://www.r-project.org/
9. The Comprehensive R Archive Network (CRAN). ://cran.r-project.org/
10. Luangkesorn, L.: Introduction to R-GLPK. Repository CRAN, October 12, 2006, http://www.cran.r-project.org/
11. Lee, L., Luangkesorn, L.: Reference manual of Package 'glpk', GNU Linear Programming Kit. Repository CRAN, April 17, 2009, Version 4.8-0.5, http://www.cran.rproject.org/
12. GAMS Data Exchange The Comprehensive R Archive Network (CRAN). http://support.gamssoftware.com/doku.php?id=gdxrrw:interfacing_gams_and_r
13. Roslyakova, I.: Modified Segmentation Methods of Quasi-stationary Time Series. useR conference 2010, http://user2010.org/slides/Roslyakova.pdf

Chapter 10
VisPlain®: Insight by Visualization

Axel Hecker and Arnd vom Hofe

Abstract Modeling tasks typically deal with complex relations between objects of different kinds. In this chapter we describe the graphical tool VisPlain® that provides visualization of such objects and their relations as well as interactive functionality for demonstration purposes and analysis. VisPlain® is meant to smoothen modeling processes by providing visual representations of them or of parts of them.

10.1 Introduction

This article describes a graphic tool whose purpose is to easily create graphical representations of data typically used in modeling. The general idea is: A modeler has to deal with data that, in most cases, come from some external system, sometimes in large volumes. It would be useful to have a tool that allows to represent those data or parts of them in such a manner that people in a meeting can easily point to their subject of discussion. VisPlain® is not a modeling tool, but a supporting tool that may be useful for all communication around modeling: It exPLAINs by VISualization.

VisPlain® visualizes complex structures and allows the user to analyze them interactively. Such complex structures may be supply chains like a production process for example, logistic scenarios like the distribution of goods via warehouses, workflow and control flow patterns like order management in a certain business, hierarchical structures like business hierarchies, investment relations etc.

Based on a standard and easy to generate file format, VisPlain® will automatically create a comprehensive and esthetically well balanced graphical representation of the provided data.

The concept of VisPlain® is simple. It expects data consisting of

A. Hecker (✉) · A. vom Hofe
Mathesis GmbH, Friedrichsplatz 11, 68165 Mannheim, Germany
e-mail: visplain.ml@mathesis.de; visplain.ml@mathesis.de

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_10, © Springer-Verlag Berlin Heidelberg 2012

- Nodes,
- Directed connections between nodes, and
- Attributes belonging to nodes or to connections.

A *node* is VisPlain®'s base unit. Typically, it either represents a place where any kind of operation happens like production, storage, purchase, sale, resource allocation etc.; or an actual location, for example a geographical location or a position in a structure, like in a hierarchy.

Connections start at a node and point to another node, they visualize a flow. This flow may contain plenty of different things like material flow, control flow, cost flow etc. In many cases, a certain connection might represent more than one flow at once; e.g., a material flow might be associated with a certain cost flow (costs representing the value of that material, or transport costs).

Attributes are associated to nodes and connections. Any node and any connection may have a (theoretically) unlimited number of attributes. An attribute typically describes something that can be measured, e.g., a resource that is either produced or consumed, or a certain cost.

Nodes, connections, and attributes are the three basic objects used by VisPlain®. Nodes and connections are automatically arranged on each image created. Attributes are primarily subject to differentiated interactive investigations of this image. They allow the user to show and highlight details of special interest.

VisPlain®'s features can be summarized like this:

- Automatically generate a good-looking arrangement of nodes and connections.
- Provide functionality to interactively analyze nodes and connections in respect to the attributes associated to them.
- VisPlain® focuses on interactive visualization and nothing else—unlike other systems that may provide visualization as an add-on, typically bound to the use of these systems, often demanding complicated maintenance tasks and high costs.

10.2 A Simple Example

Given a simple two step production process in the chemical industry:

- Reaction, consuming two raw materials R1 and R2, producing intermediate product I1.
- Granulation, consuming I1, producing two different finished products F1 and F2.

For simplicity reasons it is assumed that raw materials come from specific raw material storage tanks and that finished products go to specific silos.

This scenario can roughly be described in terms of nodes and connections like this:

Level	Node from	Connection	Node to
1	Raw material tanks	Flow of raw materials R1-R2	Reaction
2	Reaction	Flow of intermediate product I1	Granulation
3	Granulation	Flow of finished products F1-F2	Finished product silos

This description can be transformed into a file in spreadsheet format that approximately looks like this:

From-1	From-2	To-1	To-2	Attribute	Value	Unit
Raw Materials	Raw Mat.Tank 1			R1	20	T
Raw Materials	Raw Mat.Tank 2			R2	30	T
Production	Reaction			I1	70	T
Production	Granulation			F	55	T
Final Goods	Silo 1			F1	1	T
Final Goods	Silo 2			F2	2	T
Raw Materials	Raw Mat.Tank 1	Production	Reaction	R1	10	T
Raw Materials	Raw Mat.Tank 2	Production	Reaction	R2	25	T
Production	Reaction	Production	Granulation	I1	70	T
Production	Granulation	Final Goods	Silo 1	F1	1	T
Production	Granulation	Final Goods	Silo 2	F2	2	T

Explanation:

- The data file represents a table consisting of columns (fields) and lines
- The first line provides standardized field names:
  - From-1: Node type
  - From-2: Name of a node
  - To-1: Node type: when empty, the line describes a node, otherwise, it describes a connection: From—To
  - To-2: Name of another node
  - Attribute: Name of an attribute associated to the node respectively the connection
  - Value: Quantity, price or whatever information is associated to the respective attribute
  - Unit: Unit of Measurement valid for the value given
- The other lines provide the data VisPlain® uses in order to create a graphical representation

When the given data is fed into VisPlain®, the resulting graphical representation will look like as shown in Fig. 10.1:

Explanation:

- Nodes appear as “boxes” with rounded edges, showing the node name and the name/value of their attribute.

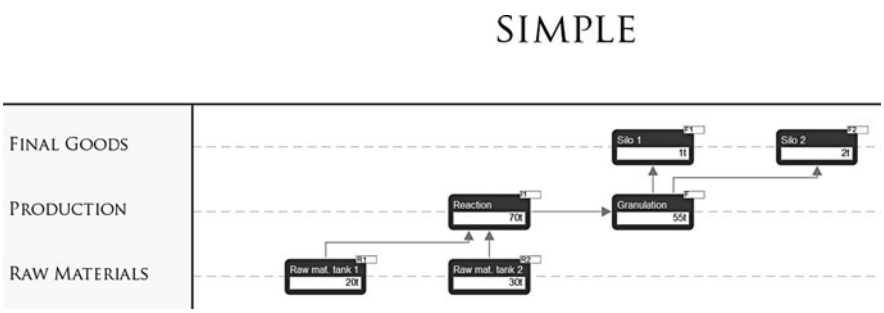


Fig. 10.1 A very simple example

- Connections appear as arrows.
- Node types appear on the left hand side, providing a grouping for nodes belonging to that type; in this simple example, nodes belonging to the same type appear on the same line.

To sum up, VisPlain® uses a simple data file that can easily be extracted from many existing ERP systems and transforms those data into a graphical representation.

10.3 A More Realistic Example

The example above was kept simple for the sake of demonstration. A more realistic example will contain a lot more information: the number of nodes or connections might be significantly larger and each node or each connection may have several attributes. Attributes for given nodes or connections can easily be added by repeating the key fields describing the node respectively the connection.

The example below comes from the refinery industry, and it describes the production process of different kinds of petrol. When fed into VisPlain®, such example may look like as shown in Fig. 10.2:

Explanation:

- This scenario shows 32 nodes and 56 connections.
- As in the simple scenario, nodes are grouped according to their level in the supply process: raw materials are provided at the bottom, leading to the first production step (“Distillation”) on the next level and so on, until end products reach final storage at the top. In this example, vertical levels are fixed—VisPlain® was not allowed to change them.
- On the contrary, in the horizontal direction, VisPlain® was allowed to arrange nodes in such a way that a comprehensive result is achieved.
- “Comprehensive” means: avoid clustering of nodes; arrange nodes in a way that connections between them are as short as possible and overlaps of connections and nodes are reduced as much as possible.

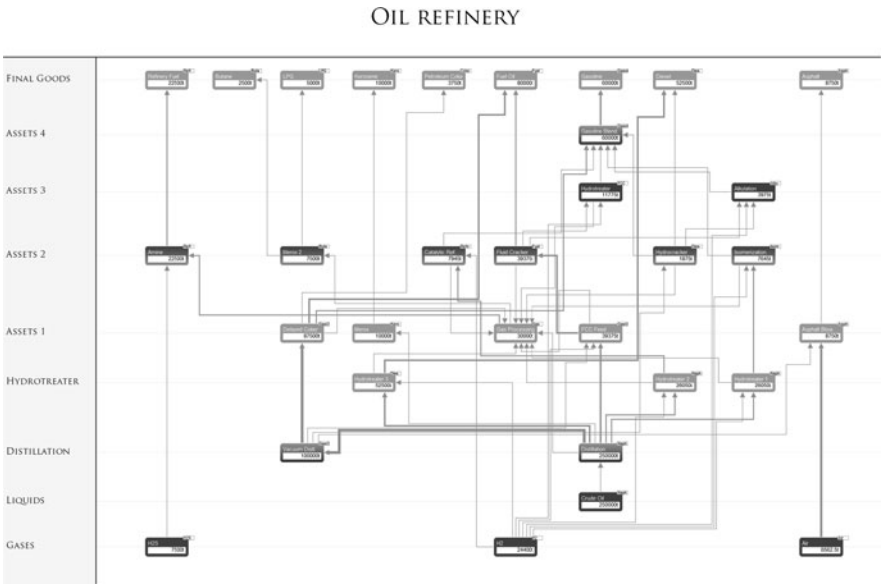


Fig. 10.2 A more realistic example

- The picture can be panned, zoomed, and printed.
- Each node has a name, a numeric value and a flag that presents the attribute name whose value is shown.
- Each connection has a thickness, representing the flow size of the primary attribute associated to that connection.

To sum up, VisPlain® is able to provide a well structured graphical representation of data described in the underlying data file; this is done automatically without any user intervention.

But this is only the beginning. As soon as a graphical representation is generated (like the one given above), the user may analyze this representation interactively.

10.3.1 Hovering Over a Node with the Mouse Pointer

As soon as the user "touches" a node by moving the mouse pointer on top of it, it will highlight the node itself and all connections associated to it (see Fig. 10.3):

In this example, highlighting is restricted to one node and connections directly associated to it. In a more complex situation, highlighted connections may affect nodes of other levels, which provoke more connections to be highlighted and so on. By this technique, an arbitrary number of nodes and connections can be visualized, many levels deep.

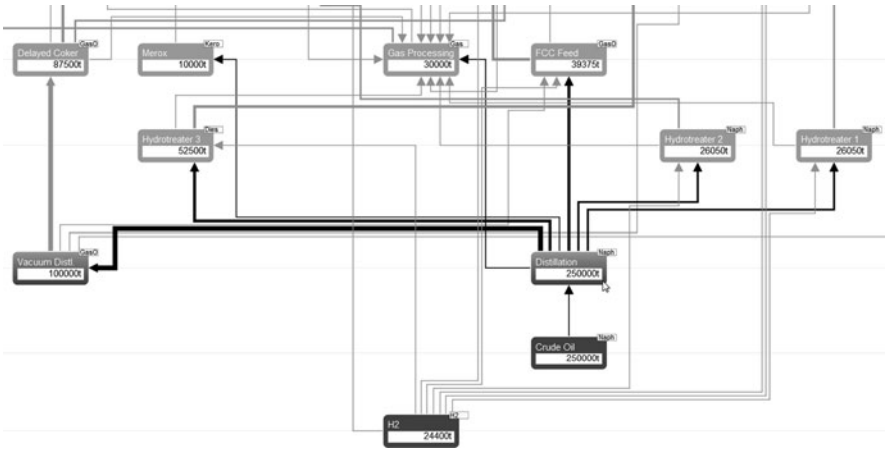


Fig. 10.3 Hovering over a node with the mouse pointer

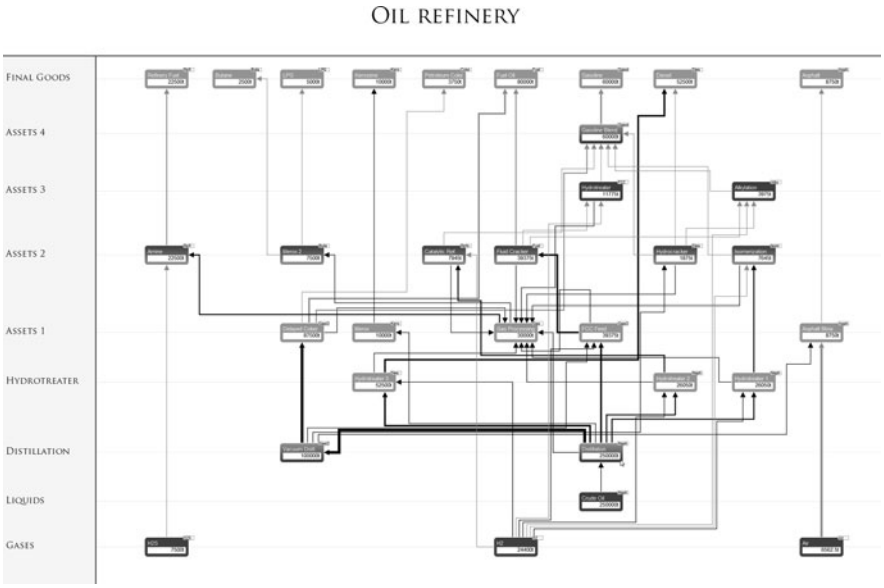


Fig. 10.4 Hovering over a node . . . two levels deep

For example, the user chooses to highlight objects two levels deep. The resulting image may look like as shown in Fig. 10.4:

To sum up, Highlighting nodes and connections by hovering with the mouse pointer over a specific node instantly shows all objects associated to it, optionally over several levels.

10.3.2 Selecting Attributes

At the top left corner of the image, VisPlain® optionally creates a legend listing all attributes that it finds in the underlying data. Each attribute can be selected individually by clicking its node (= left: green) or its connection (= right: blue) switch. Or all attributes can be selected at once, by clicking one or both of the bottom switches.

When the user selects the left (green) bottom switch, hovering over a node with the mouse pointer brings up a list of all attributes belonging to that node, with their associated values (see Fig. 10.5):

Explanation:

- In this example, node "Vacuum Distl." has four attributes: GasO (Gas Oil), RCost (Raw Material Costs), VCost (Energy Costs), DCost (Distribution Costs).
- The first attribute serves as primary attribute; it is the one whose name and value is shown per default (flag and value).
- The other attributes are secondary attributes, listing anything a user might be interested in. A typical application of secondary attributes is cost information like production costs or energy costs.

This is also applicable to connections. In this example (see Fig. 10.6), the user chose to show all connection attributes at once, without hovering over certain connections with the mouse pointer.

Both options—show all attributes / only show attributes touched with the mouse pointer—are available.

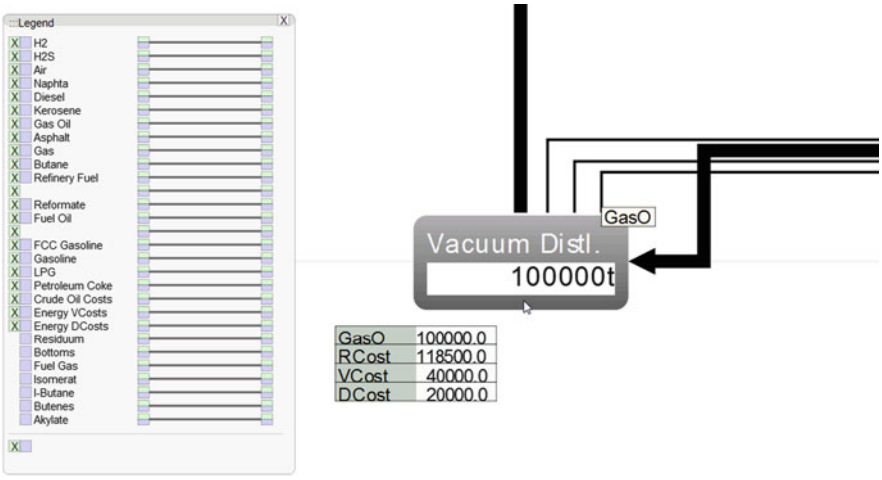


Fig. 10.5 Selecting attributes: nodes

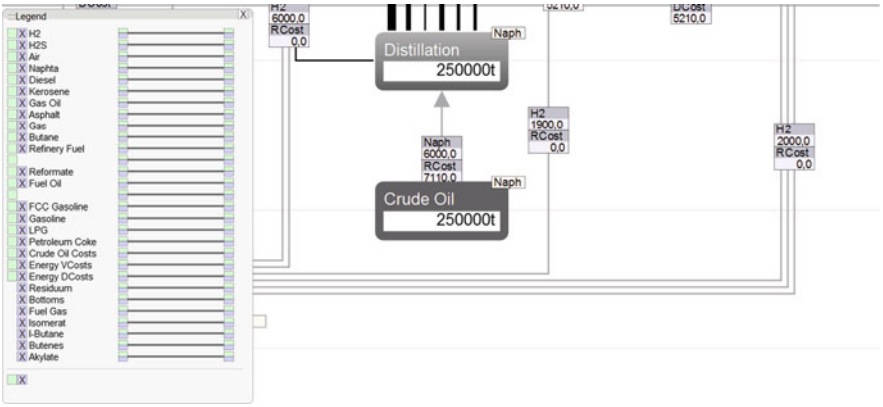


Fig. 10.6 Selecting attributes: connections

To sum up, all underlying attributes—be it attributes of nodes or attributes of connections—can be made visible in the form of attached lists associated to their respective parent objects. By various selection options, the user has full control on which attributes should appear in those lists.

10.3.3 Derived and Calculated Attributes

All attributes shown so far are explicit attributes: VisPlain® shows these attributes exactly as they appear in the underlying data file. Additionally, VisPlain® has the ability to automatically provide derived attributes and show them, as well.

What are derived attributes? In the example above, product "Naph" (Naphta) is produced by using raw material "Crude Oil". Consequently, raw material "Crude Oil" is implicitly present in this product, as well as the cost that is associated to this raw material. For instance, it might be interesting to know how much of each raw material and how much of the corresponding raw material costs are implicitly present at the very end of the chain: in finished products, as they arrive at customers.

VisPlain® automatically propagates such attributes, so that their values can be followed forwards and backwards through the whole chain. Such attributes are called "Derived Attributes": their values are automatically added to the list of node and connection attributes, such that they can be analyzed exactly the same way. This allows very powerful visual demonstrations of cost and material flows, including: cost flows, identification of profitable and weak business lines, ecological aspects like energy consumption, and more.

The following example displayed in Fig. 10.7 shows raw material (crude oil) costs for nodes representing the end of the production chain:

This ability is completed by so-called "Calculated Attributes". While Derived Attributes are always present in the underlying data—VisPlain® only propagates

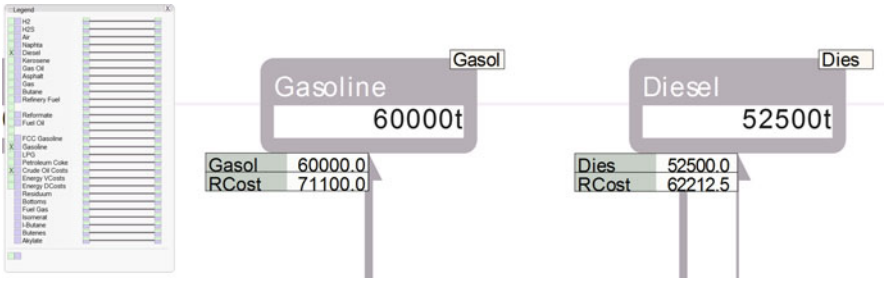


Fig. 10.7 Derived attributes

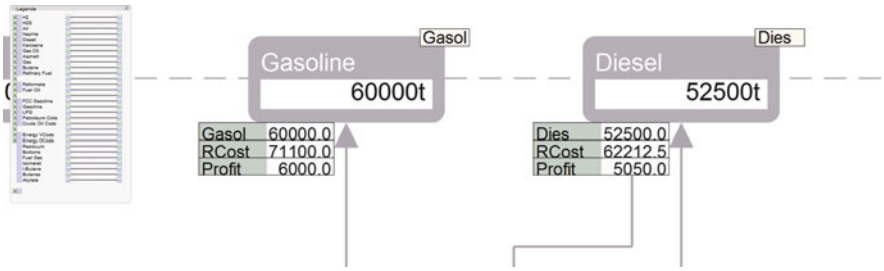


Fig. 10.8 Calculated attributes

them forwards and backwards through the chain of nodes -, Calculated Attributes do not appear in the underlying data at all. They have to be defined by the user by providing formulas how they can be calculated from other existing attributes.

Example: Finance departments might calculate profit according to the following formula: Income minus all variable costs. This formula can be input in VisPlain®'s list of Calculated Attributes, telling it to show this attribute in all cases where attributes the calculation is based on, appear as well.

Consequently, VisPlain® is able to provide, for example, a graphical representation of individual final products, each associated with calculated attributes providing respective profits. An example is shown Fig. 10.8:

In case end customers or customer groups were present in this picture, this principle could be extended, showing customers or customer groups together with according profits. Like this, a very intuitive picture of a certain business can be created, with leading figures graphically attached.

To sum up, besides attributes actually present in the underlying data, as they describe nodes and/or connections, VisPlain® has the ability to derive and calculate additional attributes and add them to the list of existing attributes. Derived Attributes are taken from existing attributes, propagating them from the spots (nodes/connections) where they actually appear forwards and backwards through the whole chain of nodes. Calculated Attributes are artificial attributes that do not appear at all in the underlying data, but are defined by calculation formulas, providing the ability to analyze the underlying data on a very general level—like "profit" for example.

10.3.4 Selecting Nodes for Reporting in Table Format

Nodes can be selected, either manually or automatically. Manual selection is done by double clicking the respective node. Figure 10.9 shows an example where a node has been manually selected:

Nodes manually selected are marked with a visual frame.

Automatic selection of nodes or connections is done by using their attributes. VisPlain®'s legend not only shows the attributes present in the current scenario, it also supplies sliders for each attribute. Sliders can be drawn from left to right or from right to left: The left slider increases a lower limit on values assigned to that attribute, the right slider decreases an upper limit in a similar way. By this, certain "windows" on values can be selected; when a node has this attribute with a value inside the window, it is marked with a shadow-like background (see Fig. 10.10):

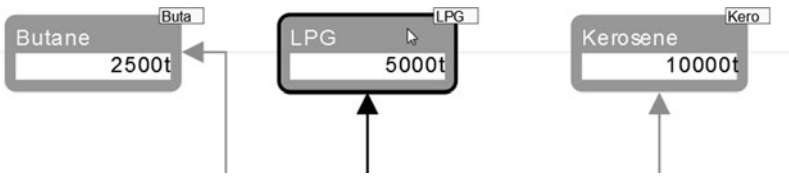


Fig. 10.9 Selecting nodes manually

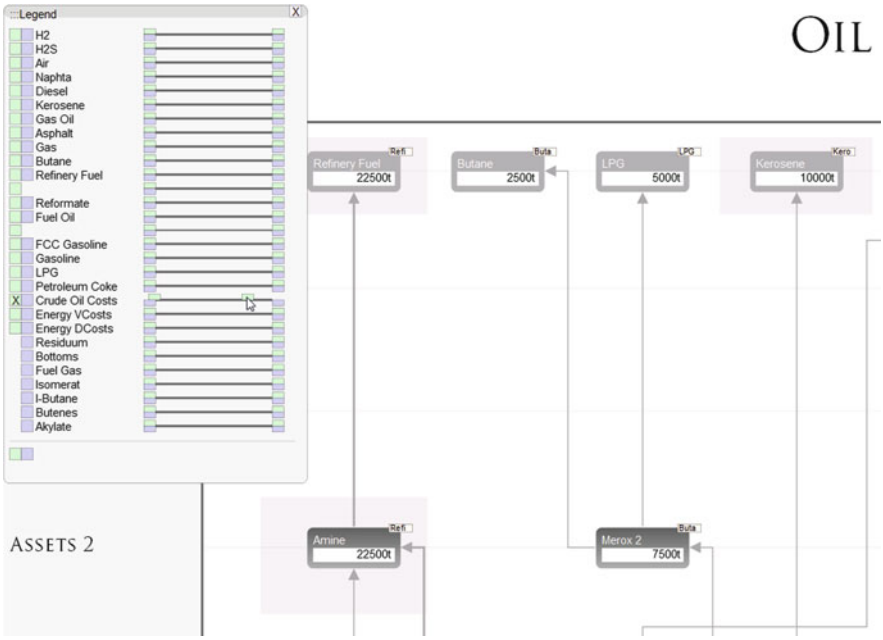


Fig. 10.10 Selecting nodes automatically

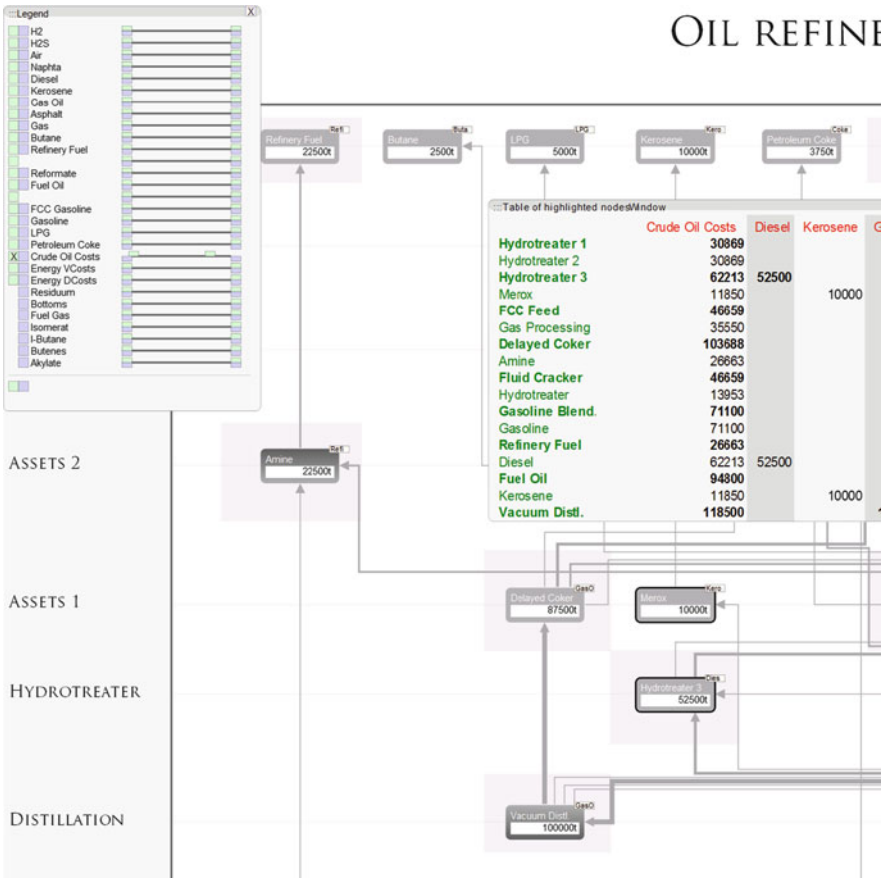


Fig. 10.11 Tabular reporting on specific nodes

Both selections—the manual one as well as the automatic one—lead to certain nodes as being selected. At this point, various operations may be executed on the selected nodes. For example, a standard (tabular) reporting can be created for these nodes (see Fig. 10.11):

This report shows the selected nodes, each with complete lists of their attributes and values.

10.4 A Complex Example

The examples shown so far were (a) a simple example that was meant to describe VisPlain®'s basic structures, and (b) a more realistic example that was meant to demonstrate VisPlain®'s features in detail.

At the other end of the scale, VisPlain® allows to deal with large scale structures, like the one displayed in Fig. 10.12, for example:

Obviously, this representation is too complex for analyzing details—it drives the abilities of VisPlain® to some extreme that may be beyond what a normal user likes to see. Nevertheless, this example has been generated exactly the same way as the other examples shown before. Consequently, all features described are present, e.g., the feature "hovering over a box"(see Fig. 10.13):

The box at the bottom is highlighted together with all its connections to other boxes.

As soon as the user zooms into this picture, all details hidden become visible again (see Fig. 10.14):

Consequently, the user can switch between the overview and any detail at will.

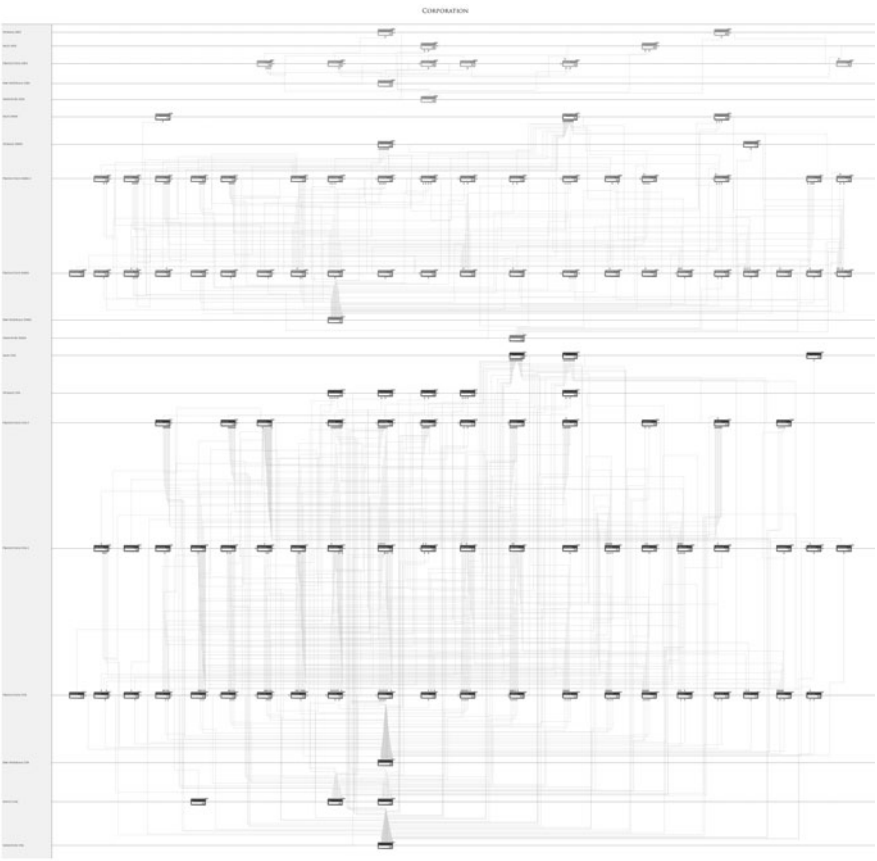


Fig. 10.12 A complex example

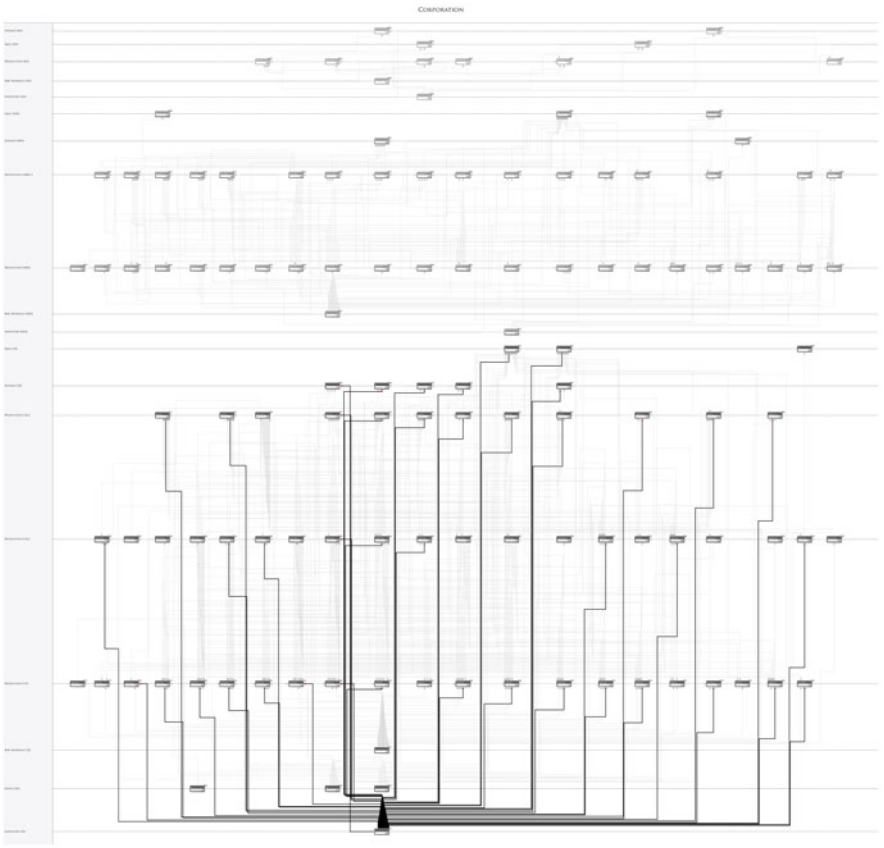


Fig. 10.13 A complex example: hovering over a box

10.5 Summary

In a modeling scenario VisPlain® supports the identification of technical and operational problems by well-structured and comprehensive data visualization. Its main purpose is improved and simplified communication in order to avoid misunderstandings, e.g., when discussing details of a modeling process. Experience shows that it is often difficult to express precisely what you mean when a heterogeneous group of people tries to clarify the concepts and details of a modeling task. The efforts laid into the development of VisPlain® try to respond to this experience. VisPlain® is a tool that can smooth the process of clarification of important details substantially.

In order to provide this functionality, VisPlain® focuses on simplicity: The data expected by VisPlain® can be laid down in an easy-to-read, easy-to-generate spreadsheet format. VisPlain® itself is used inside a common browser (Internet

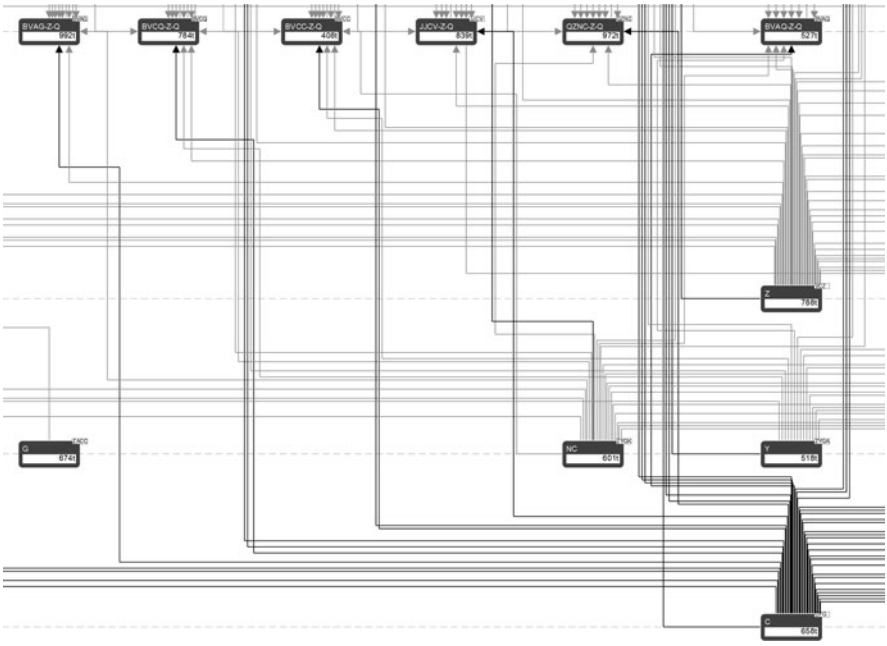


Fig. 10.14 A complex example: zoom for details

Explorer, FireFox ...); it does not have to be installed as an application on each PC using it.

VisPlain® can be used outside modeling processes as well. The ambition is to provide a general interactive tool for visualizing structures that typically arise in business life. The examples shown in this article focus on production and logistic processes. But equipped with its powerful placing algorithm, VisPlain® can demonstrate its abilities in many other contexts: hierarchical structures (like business hierarchies), network structures (like flow charts), time controlled dependencies (like Gantt charts). In principle, VisPlain® can visualize any scenario that follows the basic concepts described in this article, consisting of: nodes, directed connections between them, and attributes belonging to both of them.

Chapter 11
Economic Simulation Models in Agricultural Economics: The Current and Possible Future Role of Algebraic Modeling Languages

Wolfgang Britz and Josef Kallrath

Abstract This contribution is on the current and future role of algebraic modeling languages. While some of the discussion is based on special experience using GAMS for economic simulation models in the field of agricultural economics, other aspects are general to all modeling system, e.g., the modularization of code. Another common aspect is the transition of larger and larger modeling applications and optimization projects into IT projects leading to the open question: How far can we go with algebraic modeling systems?

11.1 Introduction

Agricultural economics are by definition a branch of applied economics, covering single firm planning to global analysis of trade and agricultural policies. Differences in scales, research focus and methodology have led to a large portfolio of different simulation models. We will for brevity in the following use the term model for simulation models, in contrast to the more general connotation used in economics. Many of the models applied in agricultural economics require specialized solvers, are based on identically structured equations and thus are possible candidates for an implementation in modeling languages. The paper aims at a systematic overview on use of algebraic modeling languages (AMLs) in agricultural economics as well as well as related advantages and disadvantages. It is organized as follows. The first section gives a short overview of major model types and how they are

W. Britz (✉)
Institute for Food and Resource Economics, University Bonn,
Nussallee 21, 53115 Bonn, Germany
e-mail: wolfgang.britz@ilr.uni-bonn.de

J. Kallrath
Department of Astronomy, University of Florida, Gainesville, FL 32611, US
e-mail: kallrath@astro.ufl.edu

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_11, © Springer-Verlag Berlin Heidelberg 2012

typically implemented in software. Next, we discuss how these economic models are integrated into the more general notion of a tool, and develop from there profiles for a successful IT integration. In the third section we discuss specifically why GAMS might be an often chosen package, before we tentatively try to look ahead at future requirements in the fourth section. Finally, we summarize and conclude.

11.2 Model Types in Agricultural Economics

11.2.1 Single Farm Models

Computer based models in agricultural economics date back to the 1950s when LP models solved on computers were used the first time to optimize single farm programs. The constraints restrict the production possibilities of the farm by defining limiting resources such as available land, labor, irrigation water, stable places or machineries as well as further technological details relating, e.g., to crop rotation restrictions, animal or crop nutrient requirements. Crop areas, herd size, investment activities or even specific processes such as ploughing are the decision variables, costs and revenues linked to them enter the objective function. The research teams focusing on farm management keep that branch of models alive, today building on fully dynamic mixed-integer models, often considering different state-of-natures. Perhaps due to their focus on single firms, there seems to be a tendency to host the modestly size data sets underlying the models in spreadsheets and to use specialized spreadsheet add-ons offering access to solvers or add-ons for Monte-Carlo simulations such as "@risk". The use of algebraic modeling languages seems to be of minor importance. Single farm models are also widely used in applications related to resource and environmental economics, typically called bio-economic models (cf. [11]). The later terms highlights that these models are linked with bio-physical process models (cf. [24]) which describe, for instance, the interaction between soil, climate, farm management, crop growth and water and the nutrient cycle. These bio-physical models are used to parameterize blocks of coefficients in the constraint matrix and to quantity effects of farm management on the environment. A recent example provides the development of a model template called FFSIM [26], realized in GAMS. Compared to MIP applications, e.g., in scheduling problems, the models are relatively small by typically featuring a few hundredths to a few thousands variables and equations. However, typically many different instances representing regions and farm types are generated based on the same template.

11.2.2 Aggregate Programming Models

In the 1970s, helped by the eased access to mainframe computers by university departments, the single farm models were applied to so-called "aggregate farms"

for policy questions. Data for these models did typically stem not from single firm observations, but rather from aggregate agricultural statistics, for instance, on land use, herd sizes, yields and number of farms. Quite often, many of these models were solved simultaneously while being linked by common constraints, leading to rather large LPs, at least at that time, with thousands of variables and equations. Consequently, specialized matrix generators were developed, typically realized in FORTRAN, which can be seen as one root of AMLs. The perhaps most prominent example provides the CARD-LP system [17] dating back to the seventies, where later releases were realized in GAMS. The CARD systems covered the whole US farm-land broken down by production regions. Similar systems were developed in many other countries (cf. [4] for a German application). LPs face typical problems when applied to aggregate of firms such as jumpy behavior to changes in shocks and a tendency to overspecialized solutions, often in sharp contrast to the observed behavior of the industry. Equally, their calibration to observed data is rather resource intensive (cf. the chapter on model calibration in [20]). Additionally, applying LPs to a whole industry and deriving policy advice from there carries a clear notion of central social planning, not well fitting to the market based solutions favored by policy makers world wide since the 1980s. The combination of the methodological shortcomings and the evolvement of new model types such as multi-commodity models (see next section) led for some years to a sharp decrease in their application. Howitt [23] introduced with Positive Mathematical Programming (PMP) a new methodology to calibrate programming models based on a quadratic cost function. That led to a branch of methodological developments overcoming the normative character of LPs by basing their simulation response on observed behavior. As a consequence, older aggregate LP models were changed to PMP and new models based on PMPs arrived in the market [21], often implemented in `GAMS`. Once the former LP community was getting used to QP/NLP applications, it quickly started embed non-linearities also in the constraints. The development was clearly helped by improvements in NLP solvers and the dramatic increase in computing power of desktops. A recent example of NLP based solutions provides the extension CAPRI [10] with about 2,000 farm type models [18], each with about 300 variables and equations. These models can be solved on a multi-core system in about 40 s. As in the case of bio-economic single farm models, GAMS seems currently to dominate the market for aggregate programming models.

11.2.3 Partial Market Models

Partial market models differ from farm and aggregate programming models by endogenous prices which require demand and trade as endogenous variables—at least indirectly—in the equation structure of the model. Note that the term endogenous variables used in economic corresponds to variables in mathematics, while exogenous corresponds to data or parameters. An extension of aggregate programming models to allow for price endogeneity consists in changing the objective

function by replacing the revenues—the product of endogenous quantities and given prices—by consumer surplus, i.e., the integral under the indirect demand curve. The latter describes the price as a function of demand quantities. As a consequence, the optimization problem now maximizes net social welfare, and not only farm profits. If linear indirect demand functions are used, a quadratic programming problem emerges. When interregional trade and related transport costs are added, a variant of a so-called Takayama-Judge [37] type model evolves. The probably best known example of the combination of aggregate programming models for different regions, transport cost minimization and price endogeneity depicted over consumer surplus in the objective function is the strand of models culminating in the FASOMGHG model [1], with roots dating back to the seventies. FASOMGHG is a dynamic, nonlinear, price endogenous, mathematical programming model covering all agricultural and wood product markets with a focus on the US. Nonlinear terms in the objective function are replaced by piece-wise linear approximations to yield a LP. Some versions of the model might comprise more than 1 million variables and equations. FASOMGHG and variants thereof are all realized in GAMS. Bruce McCarl, one of the developers of FASOMGHG, is well known in the community, not at least due to his Expanded GAMS Guide [28] and regular GAMS courses. He provides one of several examples of a fruitful co-operation between the software companies responsible for AMLs and their clients. The 1980s saw the evolvement of so-called Multi-Commodity Models (MCMs), constrained systems of equations where supply and demand enter market clearing equations and depend on endogenous prices. These models were and are widely used to analysis impacts of trade and domestic policies. Many research branches of agricultural ministries, but also international organizations such as the FAO, OECD or IFPRI maintain large-scale multi-commodity models. They are typical global in coverage, simulating simultaneously many regions and products (see [9] for a recent review with a focus on Europe). The root of one branch of MCM models was the so-called Static World Policy Simulation Model (SWOPSIM) [34]. The first versions of that type of MCMs comprised only linear relations between the natural logarithm of endogenous variables. The relation between quantities and prices, to give an example, was written as a double log function: the logarithm of the quantities depends on the sum of the logarithm of each price times an exogenous price elasticity. It was hence possible to treat the logarithm of the variables as the new endogenous terms to yield a system of linear equations, solved quite easily. That branch of MCMs is based on template equations were differences between products and regions are expressed solely by differences in parameters. They are hence especially suitable to be realized in a set-driven modeling language. Nevertheless, SWOPSIM had been developed in a spreadsheet, and it took quite a while before models of a similar structure were transferred to AMLs. A good example of a model still being relatively close to the original structure and realized in GAMS is ESIM [3]. In the same category of template based MCM models falls the CAPRI market model [10]. The latter provides examples of typical features found in modern MCM models such as strong non-linearities (e.g., a mix of linear, quadratic, logarithmic and exponential expressions, sigmoid functions or smooth

approximation of min according to Chen and Mangasarian [12] combined with a higher number of equations and variables. The current version of the CAPRI market model has a size of about 45.000×45.000, and, given a good starting point, CONOPT [16] solves that model in a few seconds on a fast desktop. However, in order to allow for a reasonable solution time in case of larger shocks, the solution process is broken down in a sequence of solves of sub-models, comprising first single commodities and later groups of commodities. These solves are in part performed in parallel based on the grid solve facility of GAMS. That process provides a good example about the use of complex scripting code in AMLs. In parallel to strictly templated MCMs, econometrically estimated single market models were combined over their cross-price effects to build a second variant of that model class. Perhaps the most prominent example for that model class is the FAPRI model [13]. As the equations for each product, region and items might differ in structure, the use of an AML is far less inviting. The FAPRI modeling system is still today realized as an application of linked spreadsheets. Alternatively, a SAS implementation exists (cf. [39]). Another well-known non-strictly templated MCM is AGLINK hosted at the OECD [30], dating back to the eighties, which is implemented today in the econometric package TROLL. A particular feature of TROLL is its ability to simulate large macro-econometric models, including forward-looking "rational expectations" models with simultaneity over time; maximum model size ranges upwards of 3,000 equations times 200 periods, using a Stacked-Time Newton algorithm [32]. A model drawing on the experiences from FAPRI for Europe called AG-Memod was first realized in spreadsheets and then transferred to GAMS. It comprises about 145.000 lines of GAMS code [15]. GAMS based MCMs are either solved as CNS (typically in CONOPT, [16]) or defined as a MCP problem (cf. [5]).

11.2.4 Computable General Equilibrium Models

Computable General Equilibrium (CGE) Models cover all input and output markets in an economy, including primary factors. They are hence clearly applied beyond agricultural economics. They are however widely used by agricultural economists, not least due to the economic weight of agriculture in developing countries. There are different "schools" which have also a history of using specific software. One originates from the Monash University, Australia, where also the GEMPACK package [19] originates from, allowing for declarative definition of a CGE. CGE models in GEMPACK are formulated as a system of differential equations in percentage change form. The solver comprised in GEMPACK is specialized for that type of constrained systems of equation. The package is used in most applications of GTAP derived models [22], a network with a rather large user group. GTAP features its own GUI (runGTAP, [31]) which builds on GEMPACK. The other two important schools use GAMS, where the one uses GAMS only code and the other one a specialized pre-processing language termed MPSGE [35] which allows a compact definition of equations for the typical functional forms used in the

CGE such as Constant Elasticity of Substitution (CES) functions. CGE are either solved as constrained system of equations without an objective in NLP solvers or based on MCP. Their equations comprise typically a mix of linear, logarithmic and exponential terms, and global application might lead to models comprising several ten thousands endogenous variables. The wide-spread use of GAMS is certainly partly due to the fact that it supports both model classes and related solvers, albeit PATH as a wide-spread MCP solver is, for instance, also supported by AMPL since 1998. There is also a clear legacy issue, as the World Bank from which GAMS emerged was also one of the first institutions building and applying CGEs for policy impact assessment.

11.2.5 General Developments and Summary

For all discussed model types, there is a tendency to expand the models, e.g., by increasing product and spatial coverage or by rendering new elements endogenous. Those developments benefited certainly by the tremendous increase in computing power of desktops, but also by continuous improvements in solvers (cf. Drud [16] for CONOPT). Equally, large-scale sensitivity analysis of models or stochastic runs are becoming popular, increasing dramatically computing needs. The expansion in model size and the generation of very large result data sets have consequences reaching far beyond questions of efficiently solving large scale models or many instances of the same model. Model results and input data which could eventually be stored and inspected in a single, small table in early versions comprise now ten thousands or even millions of non-zero data. That asks for technical solution to efficiently debug and inspect rather large, multi-dimensional and typically sparse matrices. Software packages used in modeling must therefore either provide efficient interfaces to Data Base Management Systems and/or often proprietary solutions to host large data bulks. Equally, in order to efficiently inspect and analyze the model results, results are typically aggregated, e.g., over space, time and products and indicators derived from them, asking for performing scripting abilities. And not at least, reporting in tables, graphs or maps are typical tasks related to model application, and included in model specific GUIs such as runGTAP. The discussion of the model types above underlines that even if there is a certain dominance of ALMs for certain model types, with GAMS clearly dominating the field, alternatives are widespread. Especially MCMs provide an application field where legacy and model structure lead to quite different IT implementations. Spreadsheet based solutions and implementation in econometric packages are equally common as GAMS based solutions. For some emerging model types such as Agent Based Modeling, there is clear tendency to use specialized packages [27]. The following chapter will try to analyze some common features of model types used in agricultural economics and link them to features of modeling languages in order to analyze reasons for the widespread use especially of GAMS.

11.3 From Model to Tools and Consequences for the IT Concept

The structure and resolution of economic simulation models in space and time, by product and processes is closely linked to data availability (cf. [7]). The development of models and suitable data bases is therefore often a coordinated effort. Considerable resources are invested to build suitable data bases for economic models from various raw data sets, requiring e.g. complex data balancing steps (cf. [33]). The data base work comprises, e.g., steps to map, disaggregate and aggregate data from official statistics and other sources such as questionnaires or scientific and engineering publications to the mnemonics and resolution of the economic model, to detect outliers, to fill data gaps and to merge different data sets to consistent and complete data bases. Often, part of these processing steps are realized again as optimization models where the objective is some penalty function based on differences between the original raw and consistent data. The most prominent example for such large-scale data work related to economic modeling provides the GTAP data bases [29]. Often, parameters entering the models are econometrically estimated from the data bases underlying the models. Equally, as touched upon already above, economic models are typically linked to extensive post-model processing to derive economic, social or environmental indicators. In many cases, these indicators are the quantitative results reported to the final client. Post-model processing might again comprise optimization problems such as in the spatial down-scaling step in CAPRI [25]. Finally, the different working steps need to be steered, the numerical results inspected and key findings presented in reports. The combination of data, parameters, methodological concept and the IT realization, eventually together with reporting tools, lead to complex tools which are developed and maintained over longer periods. The economic models are the core of larger tool. Consequently, the choice of a software package is geared towards efficient declaration and solution of that core. But increasingly, the package's ability to perform other working steps or at least to easily interface with other software is gaining importance. There is a certain tendency to stick to one core package for all tasks relating to a specific model. That is not astonishing as maintaining a software solution for a tool can be quite costly, especially if it is realized by combining applications realized in different packages. The use of one package for different tasks can lead to perhaps astonishing application fields. GAMS based tools might use GAMS scripting code to calculate regressions, spreadsheets might be used to host relatively large data bases and econometric package employed to solve NLPs. Clearly, alternative software implementations in these cases are often not only possible, but might be far more performing, less error prone and more elegant to realize. However, teams might consider the costs of using a whole portfolio of different software package as too high. A possible reason is the fact that many economic models are maintained, applied and further developed by rather small teams of academics, reflecting also the market size for these tools. These teams are often facing a high staff turn-over rate based on hiring non-permanent staff

such as graduate students and young post-docs for project related work. Especially permanent team members tend therefore to be generalists: they are responsible for data work, for estimation, calibration and validation, to develop new methodological solutions, to implement policy scenarios, analysis results, write up report and communicate with clients, as well as being responsible for project management. Given the impressive list of necessary skills, it is not astonishing that teams try to avoid the burden resulting from using different packages. That explains, for instance, why MCMs comprising econometrically estimated equations—where the parameter estimates are realized in a statistical package—are then also solved in statistical packages as well. Packages like SPSS, Systat, SAS, Gauss, JMP and S-Plus provide a numerical/graphical toolset. They can illustrate, they can simulate, and they can find approximate numerical solutions to numerical problems [32]. Therefore, statistical packages can often successfully cover all tasks related to a specific tool. In some cases, tools even comprising no optimization problem at all are realized in modeling languages [38], to benefit from the pre-existing knowledge of staff.

11.4 Challenges for Algebraic Model Languages

The challenges for AMLs are here rather discussed from the point of view of `GAMS` users who want to develop and support large real world applications. Other AMLs, for instance, `Mosel`, have already made significant steps in the direction of modularization and implemented the concept of distinguishing between local and global objects (data, variables, etc.). In this section we cover various topics: Language structure and modularization, GUIs and report generators, parallel execution, and where to stop?

11.4.1 Language Structure and Modularization

The widespread use of declarative languages with scripting abilities is certainly also due to the fact that many agricultural economists involved in modeling do not have a formal IT education. Simple language structures decrease their learning cost and avoid hiring specialists to code the models. Small, didactic models might even be read with a quite limited introduction to the modeling language itself. It is therefore not astonishing that many agricultural economics department use GAMS in courses at graduate level related to economic modeling to let students play with toy models. Some department even often courses focusing on GAMS only. Departments at different universities clearly benefit from using the same language: students and researchers can easier switch between teams and projects, and courses from different departments can be integrated into common programs. `GAMS` clearly supports that

process by handing out free teaching licences, providing didactic tutorials and distributing a rich library of example models.

What language features might explain the easiness by which students and researchers code their models in AMLs? Firstly, the declarative approach and its implementation in AMLs come close to formal model documentation in scientific publication. The transparent interfaces to by now rather stable solvers for different model classes reduce noticeably the required knowledge about how numerical optimization software works. Indeed, the combination of these two features gave birth to the evolvement of AMLs.

Their application is further helped by the fact that the dramatic increase in the computing power has reduced the comparative advantage to code data transformation in higher programming languages such as Fortran, C, or C++—scripting code in the modeling languages is somewhat slower, but for many practical applications in economic modeling certainly fast enough. The possible speed gain from alternative implementations might in many cases not offset the costs related to develop proprietary solution. A similar argument holds for the numerical optimization part: even highly non-linear, unfortunately scaled and larger models (at least when measured to what was considered large a decade before) can often be solved in acceptable time by using default options of solvers. As such, the combination of AMLs, advances in the algorithms underlying the solvers and the increase of computing power have dramatically changed the landscape of economic models.

Part of the success of AMLs is thus certainly due to keeping the languages simple. In GAMS, to choose the most widely use AML in the community of agricultural economic models, all symbols have global scope, and only a rather limited number of different objects is available. Any number is always presented as an element of a n-dimensional, double precision vector (which might collapse to a scalar). The concept of a subroutine or function with encapsulated, only locally visible objects does not yet exist, but is already at the horizon. String handling is rather limited. Compared to higher programming languages, these restrictions clearly reduce learning costs. They allow researchers to quickly code and debug a model, helped by integrated development environments (IDEs) delivered together with the packages which also allow inspecting quickly and efficiently results.

However, given the discussed tendency of economic models to grow and to be linked with other models, large-scale projects emerge with very large code bases, comprises ten thousands or hundred thousands of code lines. The code comprises much more then the model declaration and the solve statement—the core AMLs were developed for. The advantages of simple language features might turn into a disadvantage for large scale projects around complex tools, especially if several contributors develop independently bits of code. With all symbols being global, namespace conflicts are very likely. If languages do not support modularisation and encapsulation, more efforts compared to higher programming languages might be necessary to achieve an easy to maintain and well documented code base. If the concept of subroutines/functions/packages is missing, it becomes very cumbersome to exchange any code between projects. Whereas, for instance, agent

based modeling is relying to a large extent on open source libraries developed by the community itself, such a shared development of a common tool box is far harder to organize in AMLs which lack subroutines or local objects.

11.4.2 GUIs and Report Generators

As already touched upon above, larger models such as GTAP [31] or CAPRI [8] have developed their own, specialized GUIs. Additionally, since some years, third party products such as GSE [14] are available for GAMS with combine features of an IDE with GUI functionalities for model steerage and result exploitation. In contrast to, for instance, Mosel, CPLEX Studio by IBM, or statistical packages such as SAS which comprise GUI generators, GAMS has so far withstood the temptation to add such functionality.

Another field of increasing demand, sometimes combined with the GUI, are reporting tools to generate tables, graphs or maps from results. The modeling languages answer that demand with a mixed bag. Firstly, they invest in interfaces to Data Base Management Systems and provide interfaces to input and output in different format. Secondly, tools are added to inspect results, e.g., by allowing for pivoting, sorting and selecting such as the GDX viewer in GAMS. And, again, tools such as GSE from third parties offer functionality in that respect.

11.4.3 Parallel Execution

Over the last two decades, the dramatic reduction in the time needed to solve economic models was funneled both by ever faster processors arriving on the market and by improvements in the algorithms of the solvers. Both factors will probably play a more limited role in future. Accordingly, increased computing speed will have to come mostly from parallel computing. Various solvers such as CPLEX, GUROBI or XPRESS-OPTIMIZER already support parallel solves, and some can also be used in computing grids. However, whereas it is relatively to solve different models in parallel as necessary for MIP, much higher re-factoring efforts are necessary to efficiently compute parts of the same model in parallel or to execute scripting code in parallel.

Alternatives to the use of modeling languages such as MATLAB comprise already packages to use GPUs for massive parallel computing [36]. MATLAB is a close competitor to modeling languages as it is already in wide spread use also for economic simulation models [2]. Parallel computing is also implemented in econometric packages such as SAS (cf. [6]), another class of competing products. The increasing demand for sensitivity analysis and stochastic applications in economic models might well trigger higher demand for massive parallel applications.

11.4.4 Where to Stop?

Given larger applications and demand for more specific functionalities the question is how far should AMLs develop and compete with higher procedural programming languages such a C, C++ or FORTRAN. This is partly a technical question, a question of maintenance, and also a financial one. With a simple core structure, many customer groups can benefit a lot at reasonable cost. GAMS, for instance, has a diversity of happy customer groups in agriculture, economics, energy, and process engineering. They together provide a basis for the sustainability of the product they use, both regarding its maintenance and further development. Supply chain software often uses `Mosel` to code large-scale production planning problems. It may end up too expensive to the modeling system companies, and at the end to their customers, to fulfill all wishes and make everybody happy. The current market segmentation for AMLs might be viable as it is based on product differentiation geared towards specific applications and costumer groups. It is questionable if extensions of the different AMLs allowing the invasion into application fields of their direct competitors—other AMLs—or more indirect ones such as statistical packages or higher programming languages will increase the overall user community of AML users. They are however almost certainly increasing the combined maintenance and development costs for AMLs, and thus, in the end, might increase the cost to any user.

There are wishes brought forward by users, which are compatible with a language and it makes sense to add them, while others cannot be fit into the architectural design of the system and it is almost impossible to implement them. Another problem is caused by phase transitions when moving to larger and larger applications: Eventually, optimization projects turn into IT projects. IT people in large organization function completely different then AML users, or the problem holder. While AMLs such as `GAMS` guarantee 100% backward compatibility, the IT world has a rapid replacement cycle putting the total investment at risk. Many `GAMS` applications survived decades because all was kept internally within the `GAMS` model file and was encapsulated from the outer IT world. The POSIX utilities coming for free with GAMS add many commands to the operating system which the operating system vendors may have already terminated. The development of GDX gives another stable element to `GAMS` because it adds an independent data format to the system which makes the user becoming less dependent on the developments of the external IT world.

As such, a key strategy for AMLs in that respect might be to ease efficient interfacing to other applications. The GDX APIs in the most widely used programming languages delivered along with examples with a `GAMS` distribution provide a good example for that tactic.

11.5 Summary and Conclusion

Computer based simulation models are a widespread tool in agricultural economics, applied from farm to global level. Generally, an increase in size and an increasing use of non-linearities can be observed, while at the same time, the economic core models are expanded by post-model processing to, for instance, derive different social, economic or environmental indicators and per-model processing, for instance, to build up suitable data bases. Often, GUIs are added to steer model applications. Many of the models are strictly templated, i.e., based on identically structured equations, which renders the application of AMLS inviting with GAMS clearly being a kind of industry standard. Scripting abilities allow applying AMLs also for pre and post model processing steps. The combination of AMLs, performing MIP/NLP/MCP solvers and the increase in computing power was certainly a key factor in the tremendous development in that field, also driving methodological improvements.

However, many models in agricultural economics are by now rather complex tools of which maintenance, application and further development have the character of IT projects, where aspects such as modularization, scalability or linkages between numerical optimization, data mining and GUI development become important. AMLs are challenged by users to expand into these fields. They thus need to find a balance between two conflicting poles. At the one hand, they need to maintain their intriguing basic simplicity—the main argument for their development and difference, for instance, to libraries in Higher Programming Languages. At the other hand, they might need to add missing functionalities which are that important that users might switch to alternative implementations, but in a way which fits the basic language structure and philosophy.

References

1. Adams, D., Alig, R., McCarl, B.A., Murray, B.C.: FASOMGHG Conceptual Structure, and Specification: Documentation. Tech. rep., Texas A & M University, College Station, Texas (2005)
2. Anderson, P.L.: Business Economics and Finance with Matlab, GIS, and Simulation Models. CRC Press (2004)
3. Banse, M., Grethe, H., Nolte, S., Balkhausen, O.: European Simulation Model (ESIM): Model Documentation. Tech. Rep., University Hohenheim, Hohenheim, Germany (2007)
4. Bauersachs, F.: Ein regional und betriebsgruppenspezifisch differenziertes quantitatives Informationsund Sektoranalysesystem für den Agrarbereich (QUISS): Grundlagen und Modellaufbau. In: Bauersachs, F., Henrichsmeyer, W. (Hrsg.): Beiträge zur quantitativen Sektor- und Regionalanalyse im Agrarbereich, Agrarwirtschaft, Sonderheft **80**, S. 5592 (1979)
5. Billups, S.C., Steven, S.P., Ferris, M.C.: A Comparison of Large Scale Mixed Complementarity Problem Solvers. Comput. Optim. Appl. **7**, 3–25 (1997)
6. Bremer, R., Perez, J.S.P., Westfall, P.: Grid computing at texas techuniversity using sas. In: Proceedings of the 14th Annual South- Central SAS Users Group Regional Conference, pp. 64–72. Marquee Associates, LLC, Austian, Texas (2004)

7. Britz, W.: IT—An Unimportant Ingredient of Large Scale Models? Agrarwirtschaft **48**, 159–162 (1999)
8. Britz, W.: The Graphical User Interface for CAPRI version 2010. Tech. Rep., Institute for Food and Resource Economics, Chair of Economic and Agricultural Policy, University of Bonn, Bonn, Germany (2010)
9. Britz, W., Heckelei, T.: Recent Developments in EU Policies - Challenges for Partial Equilibrium Models. In: Proceedings of the 107th EAAE Seminar 'Modeling of Agricultural and Rural Development Policies'. January 29th - February 1st, 2008. Sevilla, Spain (2008)
10. Britz, W., Witzke, P.: CAPRI modeling documentation. Tech. Rep., University Bonn, Institute for Food and Resource Economics, Bonn, Germany (2008)
11. Brown, D.: A review of bio-economic models. Tech. Rep., Cornell University African Food Security and Natural Resource Management (CAFSNRM) Program, Ithaca, NY (2000)
12. Chen, C., Mangasarian, O.L.: A class of smoothing functions for nonlinear and mixed complimentarity problems. Comput. Optim. Appl. **5**, 97–138 (1996)
13. Devadoss, S., Westhoff, P.C., Helmar, M.D., Grundmeier, E., Skold, K.D., Meyers, W.H., Johnson, R.S.: The FAPRI Modeling System at CARD: A Documentation Summary, Technical Report 13, December 1989. Tech. Rep., Iowa State University, Food and Agricultural Policy Research Institute, Ames, Iowa (1989)
14. Dol, W.: GAMS Simulation Environment. Tech. Rep., Agricultural Economics Research Institute LEI, The Hague, The Netherlands (2006)
15. Donnellan, T., Hanrahan, K.: Impact analysis of the CAP reform on main agricultural Commodities, EOP report, 15 March 2007. Tech. Rep., Teagasc-Rural Economy Research Centre (RERC) (2007)
16. Drud, A.: On the Use of Second Order Information in GAMS/CONOPT3. In: Proceedings of the SIAM Optimization Meeting, May 2002. SIAM, Toronto (2002)
17. English, B.C., Smith, E.G., Atwood, J.D., Johnson, S.R., Oamek, G.E.: The CARD LP Model: A Documentation Summary, Staff General Research Papers 890. Tech. Rep., Iowa State University, Department of Economics, Ames, Iowa (1993)
18. Gocht, A., Britz, W.: EU-wide Farm Type Supply Models in CAPRI—How to consistently disaggregate Sector Models into Farm Type Models. J. Pol. Model. **33**, 146–67 (2010)
19. Harrison, W., Pearson, K.: Computing Solutions for Large General Equilibrium Models Using GEMPACK. Computational Economics pp. 83–127 (1996)
20. Hazell, P., Norton, R.: Mathematical Programming for Economic Analysis in Agriculture. Macmillan Publishing, New York (1986)
21. Heckelei, T., Britz, W.: Models based on positive Mathematical Programming: State of the Art and Further Expansions. In: Proceedings of the EAAE Symposium Modeling Agricultural Policies: State of the Art and New Challenges, February 3–5, 2005 (2005)
22. Hertel, T. (ed.): Global Trade Analysis: Modeling and Applications. Cambridge University Press, Cambridge (1999)
23. Howitt, R.: Positive Mathematical Programming. Am. J. Agr. Econ. **77**, 229–342 (1995)
24. Jame, Y.W., Cutforth, H.W.: Crop growth models for decision support systems. Can. J. Plant Sci. **76**, 9–19 (1996)
25. Leip, A., Marchi, G., Koeble, R., Kempen, M., Britz, W., Li, C.: Linking an economic model for European agriculture with a mechanistic model to estimate nitrogen losses from cropland soil in Europe. Biogeosciences Discuss. **4**, 2215–2278 (2007)
26. Louhichi, K., Janssen, S., Kanellopoulos, A., Li, H., Borkowski, N., Flichman, G., Hengsdijk, H., Zander, P., Blanco, M., Stokstad, G., Athanasiadis, I., Rizzoli, A., Huber, D., Heckelei, T., van Ittersum, M.: Generic Farming System Simulator. In: Brouwer, F.M., Ittersum, M.K. (eds.) Environmental and Agricultural Modeling: Integrated Approaches for Policy Impact Assessment, pp. 109–132. Springer, Dordrecht (2010)
27. Luna, F., Stefansson, B. (eds.): Economic Simulations in Swarm: Agent-Based Modeling and Object Oriented Programming. Kluwer Academic, Dordrecht and London (2000)
28. McCarl, B.A., Meeraus, A., van der Eijk, P., Bussieck, M., Dirkse, S., Steay, P.: McCarl GAMS User Guide, Version 23.3. Tech. Rep., GAMS Development Corporation, Washington D.C. (2010)

29. Narayanan, B.G., Walmsley, T.L. (eds.): Global Trade, Assistance, and Production: The GTAP 7 Data Base. Center for Global Trade Analysis, Purdue University, West Lafayette, Indiana (2008)
30. OECD: Documentation of the AGLINK-COSIMO Model. OECD/AGR/CA/APM(2006) 16/FINAL. Tech. Rep., OECD, Paris, France (2007)
31. Pearson, K., Horridge, M.: Hands-on Computing with RunGTAP and WinGEM to Introduce GTAP and GEMPACK, GTAP Resource 1683. Tech. Rep., Center for Global Trade Analysis, Purdue University, West Lafayette, Indiana (2003)
32. Renfro, C.G.: A Compendium of Existing Econometric Software Packages. J. Econ. Soc. Meas. **29**, 359–409 (2004)
33. Robinson, S., El-Said, M.: GAMS Code for Estimating a Social Accounting Matrix (SAM) using cross entropy methods. TMD discussion paper No. 64. Tech. Rep., International Food Policy Research, Institute, Trade and Macroeconomics Division, Washington DC, USA (2000)
34. Roningen, V.O., Sullivan, J., Dixit, P.: Documentation of the Static World Policy Simulation (SWOPSIM) Modeling Framework. Tech. Rep., ERS, USDA, Washington, DC (1991)
35. Rutherford, T.: Applied General Equilibrium Modeling with MPSGE as a GAMS Subsystem: An Overview of the Modeling Framework and Syntax. Comput. Econ. **14**(1–2), 1–46 (1999)
36. Sharma, G., Martin, J.: MATLAB: A Language for Parallel Computing. Int. J. Parallel Program. **37**, 3–36 (2009)
37. Takayama, T., Judge, G.G.: Spatial and temporal price allocation models. North-Holland Publishing Co., Amsterdam (1971)
38. Velthof, G.L., Oudendag, D., Witzke, H.P., Asman, W.A., Klimont, Z., Oenema, O.: Integrated Assessment of Nitrogen Losses from Agriculture in EU-27 using MITERRA-EUROPE. J. Environ. Qual. **38**, 402–417 (2009)
39. Westhoff, P., Brown, S., Binfield, J.: Why Stochastics Matter: Analyzing Farm and Biofuels Policies. In: Proceedings of the 107th EAAE Seminar 'Modeling of Agricultural and Rural Development Policies'. Sevilla, Spain (2008)

Chapter 12
A Practioner's Wish List Towards Algebraic Modeling Systems

Josef Kallrath

Abstract This final chapters summarizes the author's wishes and expectation toward algebraic modeling systems. Among this wish list are algebraic reformulations of nonlinear relations, special mathematical features, inclusion of codes, or LaTeXoutput of all relevant objects.

12.1 Introduction

Algebraic modeling languages (AMLs) and systems (AMSs) have been very open towards user input leading to improved or new functionality. If it was Christmas or a beneficent fairy would grant us a few free AML related wishes, here we list a few wishes and hopes we may find realized in future releases:

- Algebraic reformulations of certain nonlinear relations,
- Constraints defined on special sets,
- Inclusion of programming code programmed in C, Fortran, Pascal or other higher programming languages.
- Special mathematical features.
- Better support of polylithic modeling and solution approaches, and
- LaTeX output of all relevant objects.

This list may encourage other extensions.

J. Kallrath (✉)
BASF SE, Scientific Computing, GVM/S-B009, 67056 Ludwigshafen, Germany
e-mail: josef.kallrath@web.de

Department of Astronomy, University of Florida, Gainesville, FL 32611, US
e-mail: kallrath@astro.ufl.edu

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4_12, © Springer-Verlag Berlin Heidelberg 2012

12.2 Algebraic Reformulations of Nonlinear Relations

Special nonlinear functions such as $|x|$, $\max(x,y)$, $\min(x,y)$, or variable disjunction $\text{if}(\delta, x, y)$ can be reformulated as linear inequalities using an extra binary variable δ. The benefit of automatic reformulations would be that it avoids the tedious work described below. The relation $r = \max(x, y)$, for instance, can be modeled by the binary variable δ and four inequalities

$$r \geq x, \quad r \geq x, \quad r \leq x + M_x\delta, \quad r \leq y + M_y(1-\delta), \tag{12.1}$$

where M_x and M_y are some sufficiently large Big-M constants ideally being equal to the natural upper bounds on x and y. With $\max(x, y)$ being representable we can express

$$\min(x, y) = -\max(-x, -y) \tag{12.2}$$

or

$$|x| = \max(-x, x). \tag{12.3}$$

Variable disjunction $\text{if}(\delta, x, y)$ returning x for $\delta = 1$ and y for $\delta = 0$ can be modeled as

$$x - M_x(1-\delta) \leq r \leq x + M_x(1-\delta) \tag{12.4}$$

and

$$y - M_y\delta \leq r \leq y + M_y\delta. \tag{12.5}$$

Products of a binary variable δ and a continuous variable x, $r = x\delta$, or complementarity constraints, $xy = 0$, are further examples.

Such reformulations are not unique. Depending on which one is selected, a solver may perform better or worse. Therefore, the ideal way to transform a certain relation should be selected by the AML including some preferences set by the user and the capabilities of the solver. If a solver supports a certain relation or structure, the AML should only pass this structure to the solver and letting the solver deal with it. If the solver does not support it, the AML reformulates it, i.e., automatically introduces auxiliary binary variables and the appropriate inequalities.

Let us use the bilinear expression $y = x\delta$ involving a non-negative continuous variable x with upper bound X and a binary variable δ as a first example. The standard equivalent linear formulation leads to three inequalities

$$y \leq x, \quad y \leq X\delta, \quad y \geq x - X(1-\delta). \tag{12.6}$$

However, CPLEX, for instance, would perform much better if we introduce the auxiliary non-negative variable u and the equivalent formulation

$$y \leq x, \quad y \leq X\delta, \quad u = x - y, \quad u \leq X(1-\delta). \tag{12.7}$$

It would be extremely helpful, if AMLs would allow the user to enter an expression such as

$$\text{reform}(y = X * \text{delta}), \tag{12.8}$$

and would then take the appropriate reformulation and steps to communicate this to the solver. The more complicated case, $x \in [-X_-, X_+]$ with $X_-, X_+ > 0$, can be transformed as

$$y = x\delta = w - X_-\delta, \quad z = X_- + x,$$

where z fulfills $z \geq 0$, which allows us to treat $w = z\delta$ with $Z := X_- + X_+$ as shown above.

A second example, implied equalities, occurs in production scheduling. The starting time, $s_{i_2m_2}$, of task i_2 on machine m_2 is identical to the finishing time, $f_{i_1m_1}$, of task i_1 on machine m_1, if the machines and tasks are connected, i.e., if the binary variable $\delta_{i_1m_1i_2m_2}$ has value 1. One way to model this is by the using the Big-M inequalities

$$s_{i_2m_2} \leq f_{i_1m_1} + H^s_{i_2m_2}\,(1 - \delta_{i_1m_1i_2m_2}) \tag{12.9}$$

and

$$s_{i_2m_2} \geq f_{i_1m_1} - H^f_{i_2m_2}\,(1 - \delta_{i_1m_1i_2m_2})\,, \tag{12.10}$$

where $H^s_{i_2m_2}$ is a valid upper bound on the starting time $s_{i_2m_2}$, while $H^f_{i_2m_2}$ is a valid upper bound on time, $f_{i_1m_1}$. This formulation can lead to scaling problems.

A better way is to introduce non-negative continuous variables $u^p_{i_1m_1i_2m_2}$ and $u^m_{i_1m_1i_2m_2}$ and to use the formulation

$$s_{i_2m_2} = f_{i_1m_1} + u^p_{i_1m_1i_2m_2} - u^m_{i_1m_1i_2m_2}, \quad \forall\{i_1m_1i_2m_2\} \tag{12.11}$$

with the correction variables $u^p_{i_1m_1i_2m_2}$ and $u^m_{i_1m_1i_2m_2}$ subject to

$$u^p_{i_1m_1i_2m_2} \leq H^p_{i_1m_1i_2m_2}\,(1 - \delta_{i_1m_1i_2m_2})\,, \quad \forall\{i_1m_1i_2m_2 | H^p_{i_1m_1i_2m_2} > 0\} \tag{12.12}$$

and

$$u^m_{i_1m_1i_2m_2} \leq H^m_{i_1m_1i_2m_2}\,(1 - \delta_{i_1m_1i_2m_2})\,, \quad \forall\{i_1m_1i_2m_2 | H^m_{i_1m_1i_2m_2} > 0\}. \tag{12.13}$$

Note that Big-M coefficients $H^p_{i_1m_1i_2m_2}$ and $H^m_{i_1m_1i_2m_2}$ need to be sufficiently large to allow $u^p_{i_1m_1i_2m_2}$ and $u^m_{i_1m_1i_2m_2}$ to adjust for $\delta_{i_1m_1i_2m_2} = 0$ in such a way that no conflict occurs with the real equality $s_{i_2m_2} = f_{i_1m_1}$ established for $\delta_{i_1m_1i_2m_2} = 1$ enforcing that the correction variables are zero. If bounds, $F^-_{i_1m_1}$ and $F^+_{i_1m_1}$ on $f_{i_1m_1}$, and $S^-_{i_2m_2}$ and $S^+_{i_2m_2}$ on $s_{i_2m_2}$ are available, they should be exploited leading to smaller values of $H^p_{i_1m_1i_2m_2}$ and $H^m_{i_1m_1i_2m_2}$ as follows:

$$H^p_{i_1m_1i_2m_2} := S^+_{i_2m_2} - F^-_{i_1m_1}, \quad \forall\{i_1m_1i_2m_2\} \tag{12.14}$$

and

$$H^{m}_{i_1 m_1 i_2 m_2} := F^{+}_{i_1 m_1} - S^{-}_{i_2 m_2}, \quad \forall\{i_1 m_1 i_2 m_2\}. \tag{12.15}$$

An example for automatic reformulations in existing languages is available: `Mosel` provides the package *advmod*, which allows the user to state logical constraints that are automatically reformulated using indicator constraints. Within the Xpress suite, the CP solver Kalis can automatically access the MILP solver and produce a MILP equivalent formulation, for instance, of the all-different relation.

12.3 Constraints Defined on Sets

Let the set $\mathscr{S} = \{x_1, \ldots, x_s\}$ a set of s variables. An example of constraints defined on sets are special order sets of type k. Beale and Forrest [1] introduced special ordered sets of type 1 and 2 (SOS-1, SOS-2) and efficient branching schemes to exploit this structure. The constraint is that at most k elements can be different from zero, and in SOS-k the non-zero variables have to belong to k adjacent indices.

What we would like to see is to add a few more constraints missing from the mathematical programming field:

- Free, but non-zero variables,
- All-different variables, and
- Upper bounds on sum of the maximal k variables.

12.3.1 Modeling Non-Zero Variables

Let us consider a variable σ which can take all integral values between 0 and 10 but for some reason not the value 3. To represent σ, we introduce an auxiliary (continuous or integral) variable, ς, not restricted in sign with the bounds

$$-Z_1 \le \varsigma \le Z_2, \quad \varsigma \neq 0, \quad 0 < Z_1 = 3, \quad 0 < Z_2 = 7, \tag{12.16}$$

and related to σ by the equality

$$\sigma - 3 = \varsigma. \tag{12.17}$$

We call ς a non-zero variable. We can represent it in both continuous and integral cases by using a small non-negative parameter ε and two binary variables $\delta_1, \delta_2 \in \{0, 1\}$ selecting one of the disjunctive inequalities

$$\varsigma \le -\varepsilon \quad \vee \quad \varsigma \ge \varepsilon \tag{12.18}$$

or, if we want to use only $\le$ inequalities

$$\varsigma \le -\varepsilon \quad \vee \quad -\varsigma \le -\varepsilon. \tag{12.19}$$

With upper bounds $U_1 =: Z_2 + \varepsilon$ and $U_2 =: Z_1 + \varepsilon$ on $\varsigma + \varepsilon$ and $-\varsigma + \varepsilon$, resp., we get the two inequalities

$$\varsigma + \varepsilon \le U_1(1 - \delta_1), \quad -\varsigma + \varepsilon \le U_2(1 - \delta_2). \tag{12.20}$$

This leads to the constraints

$$\varsigma + (Z_2 + \varepsilon)\delta_1 \le Z_2, \quad \varsigma - (Z_1 + \varepsilon)\delta_2 \ge -Z_1, \quad \delta_1 + \delta_2 = 1 \tag{12.21}$$

or the equivalent form,

$$\varsigma + \varepsilon\delta_1 \le \delta_2 Z_2, \quad \varsigma - \varepsilon\delta_2 \ge -\delta_1 Z_1, \quad \delta_1 + \delta_2 = 1. \tag{12.22}$$

Inspection of (12.21) or (12.22) shows the validity of the implications:

$$\delta_1 = 1 \quad \Longrightarrow \delta_2 = 0, \quad -Z_1 \le \varsigma \le -\varepsilon \tag{12.23}$$

and

$$\delta_1 = 2 \quad \Longrightarrow \delta_1 = 0, \quad \varepsilon \le \varsigma \le Z_2, \tag{12.24}$$

i.e., the expected result for the non-zero variable ς. Equations (12.21) or (12.22) are algebraically equivalent so they might lead to different performance in the B&B algorithm. Therefore, in a given application it is worthwhile to try both formulations.

To give a practical example of how non-zero variables may could enter in a model consider a variable representing the temperature measured in degrees Celsius. The models involves constraints for a device that cannot operate if the temperature is close to the freezing point of water. In that case we might represent the temperature as a non-zero variable and choose $\Delta = 1°\text{C}$.

12.3.2 Modeling Sets of All-Different Elements

The *all-different relation* is useful to model problems involving a group of machine inspectors, where each of them has to check a different machine or vehicle. (We do not assume that every machine has to be visited by an inspector; there may be more machines than inspectors.) This problem can be modeled by the conventional approach using binary variables δ_{im} expressing whether inspector i inspects machine m and then adding the constraints

$$\sum_i \delta_{im} \le 1, \quad \forall m, \tag{12.25}$$

which expresses that no machine can be inspected by more than one inspector, and

$$\sum_m \delta_{im} = 1, \quad \forall i, \tag{12.26}$$

which ensures that each mechanist inspects exactly one machine.

This case was easy because we could introduce a binary variable which implemented the *all-different relation* by the inequality (12.25).

A more complicated situation arises in problems involving continuous variables $x_i \geq 0$ subject the requirement that for each pair of unequal indices $i \neq j$ the variables are also unequal, i.e., $x_i \neq x_j$. The concept of non-zero variables introduced in Sect. 12.3.1 allows us to represent the *all-different relation* by by non-zero variables ς_{ij}, the constraints

$$\varsigma_{ij} = x_i - x_j, \quad \forall i, j \mid i \neq j, \tag{12.27}$$

and the constraints (12.21) or (12.22). Let us consider the following example to illustrate the use of modeling sets with all-different elements. A company wants to produce some electronic devices. These devices have, besides others, three variables representing frequencies f_1, f_2 and f_3, which need to be chosen in order to describe the functionality of the devices. For technical reasons, low order resonances, say up to order 5, have to be avoided for the frequencies. This leads to the following relations among the continuous variables f_i :

$$f_i \neq n f_j, \quad \forall i, j \mid i \neq j, \quad n = 1, 2, 3, 4, 5. \tag{12.28}$$

We can describe this situation by the non-zero variables ς_{ij} :

$$\varsigma_{ij} = f_i - n f_j, \quad \forall i, j \mid i \neq j, \quad n = 1, 2, 3, 4, 5. \tag{12.29}$$

12.3.3 Upper Bounds on Sum of the Maximal k Variables

This type of constraints is useful if $\mathscr{S}$ represents the fractional quantities $x_i, i \in \mathscr{I}$, in an investment portfolio which can have up to I objects. We do not want to allow that the k largest portfolio members add up to more than $P\%$ of the portfolio. The problem is that we do not a priori know which variables will be the k largest ones.

To model this constraints we introduce I^2 binary assignment variables δ_{ij} which map x_i to y_j, i.e.,

$$\sum_{i \in \mathscr{I}} \delta_{ij} = 1, \quad \forall j \in \mathscr{I} \tag{12.30}$$

and

$$\sum_{j \in \mathscr{I}} \delta_{ij} = 1, \quad \forall i \in \mathscr{I}. \tag{12.31}$$

The connection of x_i to y_j is established by $2I^2$ inequalities

$$x_i \leq y_j + M(1 - \delta_{ij}), \quad \forall\{i, j\} \tag{12.32}$$

and

$$x_i \geq y_j - M(1 - \delta_{ij}), \quad \forall\{i, j\}. \tag{12.33}$$

The variables y_j are subject to the ordering inequalities

$$y_j \geq y_{j+1}, \quad \forall\, j \in \mathscr{I}. \tag{12.34}$$

Now we can easily apply the upper bound constraints on the sum onto the variables, y_j

$$\sum_{j\,|\,j \leq k} y_j \leq \frac{P}{100}. \tag{12.35}$$

As the number of binary variables growths as a function of I^2, this formulation easily reaches its performance limits if I becomes large.

Alternatively, we could generate the $\binom{I}{k} = \frac{I!}{(I-k)!k!}$ inequalities

$$x_{i_1} + x_{i_2} + \ldots + x_{i_k} \leq \frac{P}{100}, \quad \forall\{(i_1, i_2, \ldots, i_k) \in \mathscr{C}_{\mathscr{I}k}\}, \tag{12.36}$$

where $\mathscr{C}_{\mathscr{I}k}$ denotes the set of all subsets of $\mathscr{I}$ with k elements.

12.4 Inclusion of Programming Code

AMLs have their strength for declarative problem description, but when it comes to procedural bits of programming code they can become somewhat cumbersome. Therefore, it would be ideal if they supported the functionality to include programming code programmed in C, Fortran, Pascal or other higher programming languages.

First steps in this direction are visible. Xpress-Mosel users can access arbitrary C-User functions via the *Mosel Native Interface*. In Xpress-SLP it is possible to use external functions (defined in MS Excel) or internal functions (defined in Mosel) to define nonlinear constraints. `GAMS` offers the concept of *GAMS User Libraries*, which allows the user to incorporate own scalar functions (with up to 6 arguments) programmed in most common languages among them Fortran, VBA, C, or Python.

12.5 Special Mathematical Features

In this section we address obtaining all feasible combinations of discrete variables and optimal piecewise linear approximation of nonlinear functions.

Sometimes it is very useful, for a given MIP problem to get many or all feasible combinations of the discrete variables. This is useful to solve scheduling problems, which usually do not have clear objective function but are rather multi-criteria problems. This would give the planner a chance to inspects different solutions and allows him to select the best one according to his taste.

A situation, in which it would be useful to have all integer feasible solutions of a small but difficult core problem, is to use all those solutions as the basis of a column enumeration approach leading to a partition model. A similar situation occurs when solving nonconvex NLP or MINLP problems. Here one is often interested in inspecting various local maxima or minima.

Some solvers, e.g., CPLEX, XPRESS-OPTIMIZER or BARON (possibly others as well) allow to collect and show many or all solutions, but it is usually not straight forward and easy. It would help if algebraic modeling languages could support this functionality by appropriate commands.

Another special desirable features is the automatic import of optimal breakpoint systems or triangulations for piecewise linear approximation or over- and underestimators of univariate and bivariate nonlinear functions as computed by Kallrath and Rebennack [5] using direct and heuristic forwards methods. Optimal piecewise linear approximations or over- and underestimators are useful to replace nonlinear expressions in large supply network problems which are mostly MILP problems with some nonlinear expressions in additions. The computation of these optimal approximations is done a priori in models to be solved to global optimality. The integration of the optimal approximations into the supply network problem is rather cumbersome and some direct help by the modeling languages system would be of great help.

12.6 Better Support of Polylithic Modeling and Solution Approaches

Based on the Greek term *monolithos* (stone consisting of one single block) Kallrath [3] introduced the term *polylithic* for modeling and solution approaches in which mixed integer or non-convex nonlinear optimization problems are solved by tailor-made methods involving several models and/or algorithmic components, in which the solution of one model is input to another one. As discussed in detail by Kallrath [4], this can be exploited to initialize certain variables, or to provide bounds on them (problem-specific preprocessing). Mathematical examples of polylithic approaches are decomposition techniques (column generation, Branch&Price), or hybrid methods in which constructive heuristics and local search improvement

methods are coupled with exact MIP algorithms. Tailor-made polylithic solution approaches with thousands or millions of solve statements are challenges on algebraic modeling languages. Local objects and procedural structures are almost necessary. Warm-start and hot-start techniques can be essential. Especially, it would be very helpful if AMLs give the user

- Direct memory allocation access to allow for significant speed up, and
- More influence on what to keep in memory and what to through away, and
- Better support to implement own B&B (Branch&Bound), B&C (Branch&Cut), B&P (Branch&Price) algorithms in a straight forward manner.

As pointed out by Britz and Kallrath [2], this raises the question for the AML developers, how to deal with procedural functionalities and where to stop or put the border line.

12.7 LaTeX Output of all Relevant Objects

The automatic generation of a model's documentation in LaTeX would be very helpful for mathematicians, physicists, astronomers, and other communities publishing in LaTeX. This could apply to variables, data objects, constraints, but also input and output file names. The language parsers of AMLs have the relevant information. Therefore, it seems to be a doable task with the constraints being the most difficult part. It would be sufficient to transform the declarative parts of models files into an appropriate LaTeX documentation. People developing many models, i.e., researchers in the scientific community but also industrial users, would certainly appreciate this additional functionality. It would improve the documentation standard of real world problems significantly.

All here said for LaTeX could also hold for Microsoft Word, which is more the standard text processing system used in industry.

12.8 The Future

Let us conclude this chapter with some thoughts about the future of the companies forming the field. The Bad Honnef symposium formed a positive opinion that there are many new ideas around waiting for implementation, that the founders are still active and the modeling language companies provide a broad spectrum of different approaches. This pluralism is of value in itself. The market seems to support such a broad spectrum of different approaches. At least it is safe to say, this has been the case in the last decades and is still true today. Will it be so it in the future? Will some languages or modeling language providers disappear? We have already seen that Dash Optimization has been sold to FICO, while IBM purchased ILOG with their products still being mostly the same. The field has not seen mergers.

Will some modeling companies combine their languages? Some of them cover such complementary aspects that one might expect there is a good chance that they could combine both their features and customers. While users benefit strongly from the modeling languages available, the end of this book is the right place to focus also on how the modeling language providers can be supported. The answer is by using their software, communicating with them and exchanging ideas, recommending use of their software to university students, to colleagues at work or to other practitioners, and perhaps also running special sessions on modeling languages at conferences. Well organized workshops in nice alpine locations with selected participants might also be very fruitful to the field.

Acknowledgements It is a pleasure to thank Michael Bussieck (GAMS GmbH, Braunschweig, Germany), Susanne Heipcke (FICO, Marseille, France) and Steffen Rebennack (Coloroda School of Mines, Golden, CO) for their feedback and discussion on this chapter.

References

1. Beale, E., Forrest, J.: Global optimization using special ordered sets. Math. Program. **10**, 52–69 (1976)
2. Britz, W., Kallrath, J.: Economic Simulation Models in Agricultural Economics. In: Kallrath, J. (eds.) Algebraic Modeling Systems: Modeling and Solving Real World Optimization Problems. Springer, Heidelberg, Germany (2012)
3. Kallrath, J.: Combined Strategic Design *and* Operative Planning in the Process Industry. Comput. Chem. Eng. **33**, 1983–1993 (2009)
4. Kallrath, J.: Polylithic modeling and solution approaches using algebraic modeling systems. Optim. Lett. **5**, 453–466 (2011); 10.1007/s11590-011-0320-4
5. Kallrath, J., Rebennack, S.: Optimal Linear Approximations for MINLP Problems. (2012) submitted

Appendix A
Glossary

The terms in this glossary are used in the text of the book and are defined here also for the purpose of subsequent reference. Within this glossary all terms written in boldface are explained in the glossary.

Algorithm: Probably derived from the name of the Arabian mathematician al-Hwârizmî, a systematic procedure for solving a certain problem. In mathematics and computer science it is required that this procedure can be described by a finite number of unique, deterministic steps. At each step of the algorithm it is uniquely determined by the previous steps how to proceed.

ATP: Available to promise; the business process describing the confirmation of sales orders. In software systems usually a set of rules exist that allow the sales representative to respond to customer inquiries with a delivery date. Depending on the implementation and the software vendor, several variants exist: ATP looks at existing inventories and (planned) production orders, multi-level ATP considers rough-cut capacities and involves BOM explosion, and CTP (capable to promise) includes feasible scheduling of new production orders for fulfilling the new customer demand.

Basic variables: Those variables in optimization problems whose values, in non-degenerate cases, are away from their **bounds** and are uniquely determined from a system of equations.

Basis (Basic feasible solution): In an LP problem with constraints $\mathsf{A}\mathbf{x} = \mathbf{b}$ and $\mathbf{x} \geq 0$ the set of m linearly independent columns of the m x n system matrix A of an LP problem with m constraints and n variables forming a regular matrix $\mathscr{B}$. The vector $\mathbf{x}_B = \mathscr{B}^{-1}\mathbf{b}$ is called a basic solution. $\mathbf{x}_B$ is called a basic feasible solution if $\mathbf{x}_B \geq 0$.

Bill of Material (BOM): A list of components that are used in producing a material. In case the components are procured the BOM is called single-level, otherwise the BOM is multi-level.

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4, © Springer-Verlag Berlin Heidelberg 2012

Bound: Bounds on variables are special constraints. A bound involves only one variable and a constant which fixes the variable to that value, or serves as a lower or upper limit.

Branch & Bound: An implicit enumeration **algorithm** for solving combinatorial problems. A general Branch & Bound algorithm for **MILP** problems operates by solving an **LP** relaxation of the original problem and then performing a systematic search for an optimal solution among sub-problems formed by branching on a variable which is not currently at an integer value to form a sub-problem, resolving the sub-problems in a similar manner.

Branch & Cut: An **algorithm** for solving mixed integer linear programming problems which operates by solving a linear program which is a **relaxation** of the original problem and then performing a systematic search for an optimal solution by adjoining to the relaxation a series of valid constraints (cuts) which must be satisfied by the integer aspects of the problem to the relaxation, or to sub-problems generated from the relaxation, and resolving the problem or sub-problem in a similar manner.

Brownian Network Approximation: Brownian network approximations provide a means to optimize the control of queueing networks which would be far too complex to be tackled analytically at the level of the original queueing model. The naming is motivated by the fact, that a queueing process is described by one-dimensional reflected Brownian motion. Brownian network approximations are justified in the heavy traffic limit and display the important features of a control policy in sharpest relief.

Closed Machine Set: A closed machine set is a minimal set of machines from which neither load can be shifted from its inside to its outside nor in the reverse direction. Formally two machines x and y are in the same closed machines set if and only if there is a chain of job classes $z_1, \ldots, z_n$ and a set of machines $m_1, \ldots, m_{n-1}$ such that z_1 can be processed on x and m_1, z_i can be processed on $m_{\iota-1}$ and m_ι, for $\iota = 2, \ldots, n-1$, and z_n can be processed on m_{n-1} and y.

Constraint: A relationship that limits implicitly or explicitly the values of the variables in a model. Usually, constraints are formulated as inequalities or equations representing conditions imposed on a problem, but other types of relations exist, *e.g.*, set membership relations.

Constraint Programming: An alternative approach to problem solving that has been particularly successful for dealing with nonlinear constraint relations over discrete variables, such as frequently occur in scheduling and planning applications. The strength of CP lies in its use of a high-level semantics for stating the constraints that preserves the original meaning of the constraint relations.

Continuous relaxation: An optimization problem where the requirements that certain variables take integer or discrete values have been removed.

Convex region: A region in multi-dimensional space where a line segment joining any two points lying in the region remains completely in the space.

Cross Validation (CV): CV is a statistical/machine learning technique that aims to evaluate the generalizability of a classifier (or other decision) process. It does this by setting aside a portion of the data for testing, and uses the remaining data entries to produce the classifier. The testing data is subsequently used to evaluate how well the classifier works. Cross validation performs this whole process a number of times in order to estimate the true power of the classifier.

CTM: Capable-to-match; a rules-based constraint propagation planning algorithm in SAP APO taking into account production capacity, transportation relations, quotas, and priorities for computing a feasible production plan.

CTP: see **ATP**.

Cutting-planes: Additional valid inequalities that are added to **MILP** problems to improve their LP relaxation when all variables are treated as continuous variables.

Data Envelopment Analysis (DEA): DEA is an LP based technique for evaluating the efficiency of Decision Making Units (DMU). DMUs are often managed entities in the public and/or nonprofit sectors. The approach is applicable to the multiple outputs and designated inputs which are common for such DMUs. DEA is a non-parametric method where an explicit specification of the functional relationship between inputs and outputs is not needed.

Disjunctive Convex Programming: Optimization over the intersection of the union of convex sets.

Duality: A useful concept in optimization theory connecting the (primal) optimization problem and its dual.

Duality gap: For feasible points of the primal and dual optimization problem the difference between the primal and dual objective function values. In LP the duality gap of the optimal solution is zero.

Dual problem: An optimization problem closely related to the original problem which is called the primal problem. The dual of an LP problem is obtained by exchanging the objective function and the right-hand side constraint vector and transposing the constraint matrix.

Dual values: A synonym for shadow prices. The dual values are the dual variables, i.e., the variables in the dual optimization problem.

ERP: Enterprise Resource Planning systems are management information systems that integrate and automate many of the business practices associated with the operations or production aspects of a company.

Fair Queueing Conditions: Fair queueing conditions imply that different job classes have fair access to the production network. It is established through dynamic

routing of jobs in a way that machines pool under heavy traffic. As a consequence to this so-called resource pooling effect, job classes waiting times are distributed as in a system where there is a single queue for each resource pool.

Feasible point (feasible problem): A point (or vector) to an optimization problem that satisfies all constraints of the problem. (A problem for which at least one feasible point exists.)

Global Optimum: A feasible point, $\mathbf{x}^*$, to an optimization problem that gives the optimal value of the objective function $f(\mathbf{x})$. In a minimization problem, $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ holds for all other points, $\mathbf{x} \neq \mathbf{x}^*$, of the feasible region.

Goal programming: A method of formulating a multi-objective optimization problem by expressing each objective as a goal or target with a hypothetical attainment level, modeled as a constraint, and using as the objective function an expression which will minimize deviation from goals.

Heuristic solution: A feasible point of an optimization problem which is not necessarily optimal and has been found by a constructive technique which could not guarantee the optimality of the solution.

Improvement method: A method able to generate and improve **feasible points** of an optimization problem with respect to some objective function. Improvement methods cannot prove optimality, do not provide safe bounds and are not able to prove that an optimization problem is infeasible.

Infeasible problem: A problem for which no **feasible point** exists.

Integrality gap: The difference between the objective function value of the continuous relaxation of an integer, mixed integer or discrete programming problem and its optimal objective function value.

Kuhn-Tucker conditions: A generalization of the necessary and sufficient conditions for steady points in nonlinear optimization problems involving equalities and inequalities.

Linear combination: A linear combination of vectors $\mathbf{v}_1, ..., \mathbf{v}_n$ is the vector $\sum_i a_i \mathbf{v}_i$ with real valued numbers a_i. The trivial linear combination is generated by multiplying all vectors by zero and then adding them up, i.e., $a_i = 0$ for all i.

Linear function: A function $f(\mathbf{x})$ of a vector $\mathbf{x}$ with constant gradient $\nabla f(\mathbf{x}) = \mathbf{c}$. In that case $f(\mathbf{x})$ is of the form $f(\mathbf{x}) = \mathbf{c}^{\mathrm{T}}\mathbf{x} + \alpha$ for some fixed scalar α.

Linear independence: A set of vectors is linearly independent if there exists no non-trivial **linear combination** representing the zero-vector. The trivial linear combination is the only linear combination which generates the zero vector.

Linear Programming (LP): A technique to solve optimization problems containing only continuous variables appearing in linear constraints and in a linear objective function.

Local Optimum: A feasible point, $\mathbf{x}^*$, to an optimization problem that gives the optimal value of the objective function in the neighborhood of that point $\mathbf{x}^*$. In a minimization problem, we have for all other points of that neighborhood the relation $f(\mathbf{x}) \geq f(\mathbf{x}^*)$. Contrast with **Global Optimum**.

Matrix: A rectangular array of elements such as symbols or numbers arranged in rows and columns. A matrix may have associated with it operations such as addition, subtraction or multiplication, if these are valid for the matrix elements.

Mixed Integer Linear Programming (MILP): An extension to **Linear Programming** which allows the user to restrict variables to binary, integer, semi-continuous or partial-integer values.

Mixed Integer Nonlinear Programming (MINLP): A technique to solve optimization problems which allow some of the variables to take on binary, integer, semi-continuous or partial-integer values, and allow nonlinear constraints and objective functions.

Model (optimization model): A mathematical representation of a real-world problem using variables, constraints, objective functions and other mathematical objects.

Modeling system: In the context of mathematical optimization a software system for formulating an optimization problem. The optimization problem can be formulated in an algebraic language, or can be represented by a visual model. The modeling system enables the user to bring together the structure of the problem and the data, to use various solvers, to trace the values of variables, constraints, shadow prices and infeasibilities, and to display Branch-and-Bound trees.

Network: A representation of a problem as a series of points (nodes), some of which are then connected by lines or curves (arcs), which may or may not have a direction characteristic and a capacity characteristic. The network is usually represented by a graph.

Non-basic variables: Those variables in optimization problems which are independently fixed to one of their bounds.

Nonlinear function: Any function $f(\mathbf{x})$ of a vector $\mathbf{x}$ which has a non-constant gradient $\nabla f(\mathbf{x})$.

Nonlinear Programming (NLP): Optimization problems containing only continuous variables and nonlinear constraints and objective functions.

$\mathcal{NP}$ **completeness:** Characterization of how difficult it is to solve a certain class of optimization problems. The computational requirements increase exponentially with some measure of the problem size.

Objective (objective function): An expression in an optimization problem that has to be maximized or minimized.

Optimization: The process of finding the best solution, according to some criterion, to an algebraic representation of a problem.

Optimization algorithm: An **algorithm** which computes feasible points of optimization problems and proves that the best feasible point is globally optimal. For mixed integer problems, it is expected to compute feasible points and, for a minimization problem, a safe lower bound. The **simplex algorithm** and the **Branch&Bound** method are examples for optimization algorithms in **MILP**.

Optimum (optimal solution): A feasible point of an optimization problem that cannot be improved on, in terms of the objective function, without violating the constraints of the problem.

Pivot: An element in a matrix used to divide a set of other elements. In the context of solving systems of linear equations the pivot element is chosen with respect to numerical stability. In linear programming the pivot element is selected by pricing and the minimum ratio rule. In that context, the linear algebra step calculating, although not explicitly, the new basis inverse, is sometimes called the pivoting step.

Planning (production planning): Determines which amount of a product should be produced on which facility in a certain time period or time bucket. Planning uses usually discrete-time formulations and covers a time horizon of several months or quarters. It contains less operational details than **scheduling** and is mostly based on material balance relations.

Post-optimality (Post-optimal analysis): Investigation of the effect on the optimal solution of marginal changes in the problem's coefficients.

PPM: Production process model; along with the product data structure (PDS) this is the object that describes a production process including BOM information as well as the operations, activities and resources that are needed to make a certain product.

Presolve: An algorithm for use on the specification of an optimization problem prior to its solution, whereby redundant features are removed and valid additional features may be added.

Quadratic Programming (QP): Differs from LP in that there are quadratic terms in the objective function (the constraints remain linear). The decision variables may be continuous or discrete, in the latter case we speak of Mixed Integer Quadratic Programming (MIQP).

Ranging: Investigation of the limits of changes in coefficients in an optimization problem which will not fundamentally affect the optimal solution.

Reduced cost: The price (or the gain) for moving a non-basic variable away from the bound it is fixed to.

Relaxation: An optimization problem created from another where some of the constraints have been removed or weakened, or where domain restrictions of some variables have been removed or weakened.

Scaling: Reducing the variability in the size of the elements in a matrix (*e.g.*, an LP matrix) by a series of row or column operations.

Scheduling: Assigning, sequencing and timing of a set of tasks to a given set facilities. Scheduling requires much higher time resolution but is usually applied to only shorter time horizons than **planning**.

Sensitivity analysis: The analysis of how an optimal solution of an optimization problem changes if some input data of the problem are slightly changed.

Shadow price: The marginal change to the objective function value of an optimal solution of an optimization problem caused by making a marginal change to the right-hand side value of a constraint of the problem. Shadow prices are also termed **dual values**.

Simplex algorithm: Algorithm for solving LP problems that investigates vertices of polyhedra.

Slack variables: Variables with positive unit coefficients inserted into the left-hand side of $\leq$ inequalities to convert the inequalities into equalities.

Stochastic optimization: A technique to solve optimization problems in which some of the input data are random or subject to fluctuations.

Stochastic Dual Dynamic Programming (SDDP): SDDP is a decomposition algorithm for multi-stage stochastic linear programming problems. It is a row generation approach and builds supporting hyperplanes, similar to the cuts of a Benders decomposition, to approximate a future cost function. The original algorithm needs a stage-wise independent discrete distribution of the random variable, but variants exist.

Successive Linear Programming (SLP): Algorithm for solving NLP problems containing a modest number of nonlinear terms in constraints and objective function.

Surplus variables: Variables with negative unit coefficients inserted into the left-hand side of $\geq$ inequalities to convert the inequalities into equalities.

Unbounded problem: A problem in which no optimal solution exists (the objective function tends to increase to plus infinity or to decrease to minus infinity) because the feasible region is not bounded.

Variable: An algebraic object used to represent a decision or other varying quantity. Variables are also called “unknowns” or just “columns”.

Vector: A single-row or single-column matrix.

Index

J. Kallrath, *Algebraic Modeling Systems*, Applied Optimization 104,
DOI 10.1007/978-3-642-23592-4, © Springer-Verlag Berlin Heidelberg 2012