

Mathematical Modeling Basics

Tony Hürlimann

Department of Informatics
University of Fribourg
CH – 1700 Fribourg (Switzerland)
tony.huerlimann@unifr.ch

October 9, 2024
First Edition

Copyright © 2024
First Edition
All rights reserved. Department of Informatics
University of Fribourg
CH-1700 Fribourg Switzerland
Email: tony.huerlimann@unifr.ch
www: <https://matmod.ch>

*This book is dedicated to **Giulia** born today*

CONTENTS

1	What is modeling?	5
1.1	Introduction	6
1.1.1	Models and their Functions	7
1.1.2	The Advent of the Computer	9
1.1.3	New Scientific Branches Emerge	10
1.1.4	The Consequences	12
1.2	What is a Model?	13
1.2.1	Modeling – an Informal Definition	14
1.2.2	Mathematical Models	18
1.2.3	Related Concepts	23
2	How to model?	27
2.1	Introduction	28
2.1.1	Recognition and Specification of the Problem	29
2.1.2	Formulation of the (Mathematical) Model	30
2.1.3	Solving the Mathematical Model	31
2.1.4	Validation and Interpretation	32
2.1.5	The Modeling Life-Cycle	33
2.2	The Problems	34
2.2.1	Problem 1: Seven Digits Puzzle	35
2.2.2	Problem 2: The Clock Puzzle	35
2.2.3	Problem 3: The 6-Knights Puzzle	35
2.2.4	Problem 4: The 3-jug Problem	36
2.2.5	Problem 5: The Fake Coin Puzzle I	36
2.2.6	Problem 6: The Fake Coin Puzzle II	36
2.2.7	Problem 7: The Horse Race Puzzle	36
2.2.8	Problem 8: Adding Numbers	36
2.2.9	Problem 9: 3d-Diagonal	37

2.2.10	Problem 10: Square Inside a Triangle	37
2.2.11	Problem 11: Birthday Paradox	37
2.2.12	Problem 12: Length of Loch Ness Monster	37
2.2.13	Problem 13: Price of the Ball	37
2.2.14	Problem 14: Postman Problem	37
2.2.15	Problem 15: One Unit Missing	38
2.2.16	Problem 16: How old am I?	38
2.2.17	Problem 17: 7-Digits Puzzle (redo)	38
2.2.18	Problem 18: The Clock Puzzle (redo)	38
2.3	The Solutions	38
2.3.1	Problem 1: Seven Digits Puzzle	39
2.3.2	Problem 2: The Clock Puzzle	39
2.3.3	Problem 3: The 6-Knights Puzzle	40
2.3.4	Problem 4: The 3-jug Problem	41
2.3.5	Problem 5: The Fake Coin Puzzle I	43
2.3.6	Problem 6: The Fake Coin Puzzle II	44
2.3.7	Problem 7: The Horse Race Puzzle	44
2.3.8	Problem 8: Adding Numbers	47
2.3.9	Problem 9: 3d-Diagonal	48
2.3.10	Problem 10: Square Inside a Triangle	48
2.3.11	Problem 11: Birthday Paradox	49
2.3.12	Problem 12: Length of Loch Ness Monster	50
2.3.13	Problem 13: Price of the Ball	50
2.3.14	Problem 14: Postman Problem	51
2.3.15	Problem 15: One Unit Missing	52
2.3.16	Problem 16: How old am I?	53
2.3.17	Problem 17: 7 Digits Puzzle (redo)	54
2.3.18	Problem 18: The Clock Puzzle (redo)	55
2.4	Conclusion	55
3	The Modeling Language LPL	57
3.1	Introduction	58
3.2	The Basics	59
3.3	Sub-models	64
3.4	Drawings	65
3.5	Logical and Integer Modeling	66
3.6	Conclusion	66
4	Index Notation in Mathematics	67
4.1	Introduction	68
4.2	Definitions and Notations	71
4.2.1	Exercises in the Sigma Notation	71
4.2.2	Formal Definition	75
4.2.3	Extensions	79
4.2.4	Exercise 1:	85
4.3	Index Notation in LPL	88

4.4	A Complete Problem: Exercise 2	93
4.5	Data for Exercise 2	96
4.6	An Optimizing Problem Version	98
4.6.1	Version 1: The Easy Problem	98
4.6.2	Version 2: Limit Capacity	101
4.6.3	Version 3: With Setup Costs	103
4.7	Conclusion	104
5	Various Modeling Tools	105
5.1	Introduction	106
5.2	(Algebraic) Modeling Languages	109
5.2.1	GAMS	109
5.2.2	AMPL	114
5.2.3	LINDO/LINGO	121
5.2.4	HEXALY (former LocalSolver)	125
5.2.5	MiniZinc	131
5.2.6	AIMMS	137
5.2.7	MOSEL	141
5.2.8	OPL	145
5.2.9	MATLAB	148
5.2.10	Other Modeling Language Tools	151
5.3	Programming Languages	152
5.3.1	Gurobipy (Python – commercial)	153
5.3.2	OR-tools (Google)	160
5.3.3	Pyomo (Python – open source)	163
5.3.4	Gecco / APMonitor (Python – open source)	170
5.3.5	PuLP (Python – open source)	176
5.3.6	Python-MIP (Python – open source)	180
5.3.7	JuMP (Julia – open source)	183
5.3.8	Other Programming Languages	192
5.4	Further Tools	195
5.5	Conclusion	196
6	Various Model Types	197
6.1	Introduction	198
6.2	Variations	201
6.3	A Linear Program (examp-lp)	206
6.4	A Integer Linear Program (examp-ip)	211
6.5	A 0-1 Integer Program (examp-ip01)	215
6.6	An LP-relaxation of the 0-1 Program (examp-ip01r)	219
6.7	A Quadratic Convex Program (examp-qp)	224
6.8	A 0-1-Quadratic Program (examp-qp01)	228
6.9	Second-Order Cone (socp1)	231
6.10	Rotated second-order Cone (socp2)	232
6.11	A NQCP model (bilinear)	233
6.12	Largest Empty Rectangle (iNCQP) (quadrect)	234

6.13	A NLP (non-linear) Model (chain)	238
6.14	Discrete Dynamic System (foxrabbit)	241
6.15	A Simple Permutation Model (examp-tsp)	243
6.16	Capacitated Vehicle Routing Problem (examp-cvrp)	247
6.17	Binpacking (examp-binpack)	251
6.17.1	Permutation Problems as Implemented in LPL	252
6.18	Additional Variable Types	255
6.18.1	Semi-continuous Variable (semi-1)	255
6.18.2	Semi-integer Variable (semi-2)	256
6.18.3	A Multiple Choice Variable (mchoice-3)	257
6.19	Additional Constraint Types	258
6.19.1	Sos1 Constraint (sos-1)	258
6.19.2	Sos2 Constraint (sos-2)	259
6.19.3	MPEC Model Type (compl-3)	260
6.20	Global Constraints	261
6.20.1	Alldiff and alldiff	262
6.20.2	Element	263
6.20.3	Occurrence	264
6.20.4	Sequence_total	265
6.21	Conclusion	266
7	Datacubes and Pivot-Tables	267
7.1	Introduction	268
7.2	Definition of Datacube	270
7.3	Operations on Datacubes	272
7.3.1	Slicing	272
7.3.2	Dicing	273
7.3.3	Sizing	274
7.3.4	Rising	276
7.3.5	Selecting and Ordering	277
7.4	Interpretation of the Operations in the Light of OLAP	278
7.5	Pivot-Table: 2-dimensional Representation of Datacube	282
7.6	Spreadsheets	289
7.7	Conclusion	293
8	Logical Modeling	295
8.1	Introduction	296
8.2	The General Model and its Operators	298
8.3	Definitions and Logical Identities	302
8.4	Translating Boolean Expressions	305
8.5	Translating Mixed Expressions	307
8.5.1	Some Examples	311
8.6	Translation Procedure	313
8.7	Conclusion	320
	Appendix: Several Model Examples	320
8.8	Satisfiability I (sat1)	321

8.9	Satisfiability II (sat2)	326
8.10	Satisfiability III (sat3)	328
8.11	Satisfiability IV (sat4)	330
8.12	Satisfiability V (Horn Clauses) (sat5)	332
8.13	Chemical Synthesis (sat6)	334
8.14	Simple expressions (logic0)	336
8.15	Boolean Expressions (logic1)	342
8.16	Math-Logic Expression III (logic2)	347
8.17	Indexed Boolean Expression (logic3)	352
8.18	Disjunctive Constraint (logic4)	355
8.19	Two Liquid Containers Problem (logic5)	357
8.20	Logical Conditions in an LP (logic6)	362
8.21	Math-Logic Expression II (logic7)	364
8.22	Transformations of logical statements (logic8)	366
8.23	Importing Energy (import)	368
8.24	Assembling a Radio (radio)	371
8.25	Pigeonhole Problem (pihole)	373
9	My Vision	375
9.1	Introduction	376
9.2	Algorithmic Knowledge	377
9.3	Mathematical Knowledge	383
9.4	Historical Notes	385
9.5	Combine mathematical and algorithmic knowledge	388
9.5.1	The Sorting Problem	388
9.5.2	The 0-1 Knapsack Problem	389
9.5.3	The Two-Persons Zero-Sum Game	390
9.5.4	The Cutting Stock Problem	390
9.5.5	The Traveling Salesman Problem (TSP)	392
9.6	Requirements	393
9.7	Conclusion	396

LIST OF FIGURES

1.1	El Torro (P. Picasso, 1946)	15
1.2	The Intersection Problem	15
1.3	Topological Deformation	16
1.4	Solution to the Intersection Problem	16
1.5	Geometric Solution	20
1.6	Solution of the can problem	22
2.1	The Modeling-Life-Cycle	34
2.2	Transform the Problem to a Graph [43]	41
2.3	Simplifying a Graph [43]	41
2.4	Solution of the 3-Jug Problem	43
2.5	8-coins Puzzle Solved	44
2.6	12-coins Puzzle Solved [43]	45
2.7	Ordering of the Groups	46
2.8	Ordering of the Groups	46
2.9	Two Lines of Numbers	48
2.10	Probability P(n)	50
2.11	A Figure for the Problem	54
5.1	2 × 9 Queens can be placed maximally	113
5.2	Optimal Solution of "A-n32-k5.vrp"	130
5.3	A Light Up Puzzle	134
5.4	Solution of the Lightup Puzzle	137
5.5	A Sudoku Instance	177
5.6	Optimal Solution of the Facility Location	184
5.7	An example of Urban Planning	187
6.1	Geometric representation of the model	201

6.2	Largest Empty Rectangle with 50 points	234
6.3	A Hanging Chain	240
6.4	Fox-Rabbit Population over time	242
6.5	The Optimal Solution of a 30 Location Problem	246
6.6	Optimal Solution to the “A-n32-k5.vrp” Instance	250
7.1	A 3-dimensional cube	271
7.2	Slicing a Cube	273
7.3	Dicing a Cube	274
7.4	Dicing a Cube	275
7.5	A Aggregated Cube (Summation)	275
7.6	Lattice of All Aggregated Cubes	276
7.7	All Aggregated Cubes	276
7.8	Rising Identical Cubes	277
7.9	Rising Cubes by Extending and Merging Dimensions	277
7.10	Rising Cubes by Extending and Merging Dimensions	278
7.11	Roll-Op/Drill-Down Operations	279
7.12	Horizontal Nesting Display	280
7.13	Vertical Nesting Display	281
7.14	Pivot Tables by Switching Dimensions Horizontally/Vertically	283
7.15	Pivot Tables by Permuting Dimensions	284
7.16	Two Aggregated Cubes	284
7.17	A Pivot Table with Aggregated Cubes	286
7.18	A Pivot Table with Aggregated Cubes	286
7.19	Unformatted Pivot Table	287
7.20	Formatted Pivot Table	287
8.1	Syntax Trees I	314
8.2	Syntax Trees II	315
8.3	Cut-off a Subtree	315
8.4	Cut-off a Boolean Subtree	316
8.5	The Unit Cube with the 4 Feasible Points of C1	348
8.6	The Unit Cube with the 4 Feasible Points of D1	349
8.7	Four corner points	350
8.8	A Disjunctive Space	355
8.9	The Feasible Space of the Liquid Container Problem	357

LIST OF TABLES

8.1	Operators and operands in an expression	300
8.2	Indexed Operators in an expression	301
8.3	The Boolean Operators	302
8.4	The Boolean Operators as Mathematical inequalities	305
8.5	Logic-mathematical Equivalences	306
8.6	T0-Rules	316
8.7	T1-Rules	317
8.8	T2-Rules	318
8.9	T3-Rules	319
9.1	Programming Paradigms	388

PREFACE

“Each problem that I solved became a rule which served afterwards to solve other problems.”

— Descartes

“If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is.”

— John von Neumann

Modeling is becoming a more and more important discipline. In all branches of economical behavior we need to understand and formalize processes in order to use computers. Modeling is an art, but an art that can be learnt.

This book is a collection of nine papers that I have written about mathematical modeling in the last years. The *first Chapter* gives an informal and a formal introduction to the concepts of “modeling” and “model”. It explains the importance and functions of models. Certainly, “model” is a very general term, and its meaning varies throughout various branches of science (Science is modeling). Since we are interested basically in “mathematical models”, a precise definition is given. The *second Chapter* is a short introduction on the modeling process, explains very briefly its stages in order to learn how to model. A list of 18 easy problems is given to exercise the model building process. *Chapter three* is a short introduction to my modeling language LPL and gives some highlights of this language. *Chapter four* explains indexing notation in mathematics and its implementation in the modeling language LPL. Indexing is one of the most fundamental concept in mathematical notation. It is extremely powerful and allows the modeler to concisely formulate large and

complex mathematical model. Finally, some exercises in indexing are given. The goal of the *five Chapter* is to give a first impression of various modeling tools – free and commercial – that could be used to build mathematical models. The presentation for each tool is done by a small problem example which is implemented. In this way, the reader gets a first and better idea of the tool. Algebraic modeling language, Python and Julia packages are presented. *Chapter six* classifies the models into groups using various criteria: discrete, continuous, convex/non-convex, linear, quadratic, non-linear, permutation problems, and gives a concrete model and its implementation. In *Chapter seven* it is argued that pivot table is a powerful means to represent structured, multi-dimensional data on a two-dimensional space. They enable a user to show a large amount of structured data from different angles and perspectives. Pivot tables are an integrated part of the LPL modeling environment. *Chapter eight* shows how logical and Boolean modeling can be seen as an integrated part of mathematical modeling. Procedures are given on how to translate logical (also indexed) expressions into mathematical linear constraints. These are all part of the LPL system. Finally, *Chapter nine* is a personal vision on how a modeling language should be and what requirements it should fulfill. It is the result of my experiences as a teacher in operations research and as a consultant of various projects for large and small companies which used mathematical modeling.

The book is for all persons eager to learn mathematical modeling, especially for students in operations research and management sciences or business analytics, in order to solve all kind of problems. I also have books with a huge number of various problems with which one may learn modeling with concrete applications and puzzles (see <https://matmod.ch>).

The book is also for the modeling language designer, to illustrate – what I have found to be – the most important concepts in modeling and their implementations. I believe that one of the most important aspects of a modeling language is to be able to create models that are readable and compact and, of course, models that can be broken into smaller pieces of information that encapsulate related elements

To use the LPL modeling system, the reader does not need to install any software to run and solve models presented in this and other books. Only an Internet browser and an Internet connection is needed. In the electronic version of this book, a link appears in the title of each case-study. Click on it, and you are on the Internet site where the model is prompted in a text box. You can modify it interactively. Clicking the button “Send” on the Web page, sends the model to the LPL-server, runs it and returns the result after it has been solved in a new browser page. If you know well the syntax of LPL, you can even write an entirely new problem and present it to the LPL-server as well. Just click “Send” to solve it.

The book was compiled using the typesetting system L^AT_EX. However, the documentation of all “case studies” was written in LPL’s own “literate documentation system”. LPL contains a documentation tool (similar to *javadoc* for Java programs) to automatically translate a documented model into a full L^AT_EX or HTML code. Additional information for this tool can be found in the Reference Manual of LPL.

Availability of the LPL-Modeling-System

The mathematical modeling and all case studies in this book are based on the software LPL. It is my own contribution to the field of computer-based mathematical modeling.

LPL – the software – together with a documentation containing the Reference Manual, several papers and examples can be downloaded at:

<https://matmod.ch>

An electronic version of this book can be downloaded at [Modelbook](#).

CHAPTER 1

WHAT IS MODELING?

“A journey of thousand miles begins with a single step.”

Whereas mathematical solution and visualization techniques are omnipresent, modeling techniques – translating a problem into formal notation – received only little attention, although modeling is one of the most creative and mostly quite challenging task. It is an art and a discipline on its own right, and deserves a outstanding place when learning how to solve problems.

This paper is the first of several papers on modeling and it gives an informal and a formal introduction to the concepts of *modeling* and *model*. It explains the importance and functions of models. Certainly, “model” is a very general term, an its meaning varies throughout various branches of science (**Science is modeling**). Since we are interested basically in *mathematical models*, a precise definition is given. An intuitive notion is given in this paper of some basic concepts, such as *Boolean and arithmetic operators, sets, vectors, tuples, vector spaces*. However, to understand fully the mathematical examples, the reader should be familiar with these concepts. More is not needed at this point. Many more mathematical models can be found in my books, especially my puzzle book: See [My Modeling Books](#).

1.1. Introduction

Observation is the ultimate basis for our understanding of the world around us. But observation alone only gives information about particular events; it provides little help for dealing with new situations. Our ability and aptitude to recognize similarities and patterns in different events, to distil the important factors for a specific purpose, and to generalize our experience enables us to operate effectively in new environments. The result of this skill is *knowledge*, an essential resource for any intelligent agent.

Knowledge varies in sophistication from simple classification to understanding and comes in the form of principles and models. A *principle* is simply a general assertion and is expressed in a variety of ways ranging from saws, slogans, opinions to mathematical equations. They can vary in their validity and their precision. A *model* is, roughly speaking, an analogy or a mapping for a certain object, process, or phenomenon of interest. It is used to explain, to predict, or to control an event or a process. For example, a miniature replica of a car, placed in a wind tunnel, allows us to predict the air resistance or air eddy of a real car; a globe of the world allows us to estimate distances between locations; a graph consisting of nodes (corresponding to locations) and edges (corresponding to streets between the locations) enables us to find the shortest path between any two locations without actually driving between them; and a differential equation system enables us to balance an inverted pendulum by calculating at short intervals the speed and direction of the car on which the pendulum is fixed.

A model is a powerful means of structuring knowledge, of presenting information in an easily assimilated and concise form, of providing a convenient

method for performing certain computations, of investigating and predicting new events. The ultimate goal is to make decisions, to control our environment, to predict events, or just to explain a phenomenon.

1.1.1 Models and their Functions

Models can be classified in several ways. Their characteristics vary according to different dimensions: function, explicitness, relevance, formalization. They are used in scientific theories or in a more pragmatic context. Here are some examples classified by function, but also varying in other aspects. They illustrate the countless multitude of models and their importance in our life.

Models can *explain phenomena*. Einstein's special relativity explains the Michelson-Morley experiment of 1887 in a marvelously simple way and overruled the ether model in physics. Economists introduced the IS-LM or rational expectation models to describe a macro-economical equilibrium. Biologists build mathematical growth models to explain and describe the development of populations. Modern cosmologists use the big-bang model to explain the origin of our universe, etc.

There are also models *to control our environment*. A human operator, e.g., controls the heat process in a kiln by opening and closing several valves. He or she knows how to do this thanks to a learned pattern (model); this pattern could be formulated as a list of instructions as follows: "IF the flame is bluish at the entry port, THEN open valve 34 slightly". The model is not normally explicitly described, but it was learnt implicitly from another operator and maybe improved, through trial and error, by the operator herself. The resulting experience and know-how is sometimes difficult to put into words; it is a kind of tacit knowledge. Nevertheless one could say that the operator acts on the basis of a model she has in mind.

On the other hand, the procedure for aeroplane maintenance, according to a detailed checklist, is thoroughly *explicit*. The model is possibly a huge guide that instructs the maintenance staff on how to proceed in each and every situation.

Chemical processes can be controlled and explained using complex mathematical models. They often contain a set of differential equations which are difficult to solve. These models are also explicit and written in a *formalized language*.

Other models are used to control a social environment and often contain *normative* components. Brokers often try, with more or less success, to use guidelines and principles such as: "the FED publicizes a high government deficit provision, the dollar will come under pressure, so sell immediately". Such guidelines often don't have their roots in a sophisticated economic theory;

they just prove true because many follow them. Many models in social processes are of that type. We all follow certain principles, rules, standards, or maxims which control or influence our behavior.

Still other models constitute the basis *for making decisions*. The famous waterfall model in the software development cycle says that the implementation of a new software has to proceed in stages: analysis, specification, implementation, installation and maintenance. It gives software developers a general idea of how to proceed when writing complex software and offers a rudimentary tool to help them decide in which order the tasks should be done. It does not say anything about how long the software team have to remain at any given stage, nor what they should do if earlier tasks have to be revised: It represents a *rule-of-thumb*.

An example of a more *formal* and complex decision-making-model would be a mathematical production model consisting typically of thousands of constraints and variables as used in the petroleum industry to decide how and in what quantities to transform crude oil into petrol and fuel. The constraints – written as mathematical equations (or inequalities) – are the capacity limitations, the availability of raw materials, etc. The variables are the unknown quantities of the various intermediate and end products to be produced. The goal is to assign numerical values to the variables so that the cost are minimized, profit is maximized or some other goals are attained.

Both models are tools in the hand of an intelligent agent and guide her in her activities and support him in his decisions. The two models are very different in form and expression; the waterfall model contains only an informal list of actions to be taken, the production model, on the other hand, is a highly sophisticated mathematical model with thousands of variables which needs to be solved by a computer. But the degree of formality or complexity is not necessarily an indication of the “usefulness” of the model, although a more formal model should normally be more precise, more concise, and more consistent. Verbal and pictorial models, on the other hand, give only a crude view of the real situation.

Models may or may not be *pertinent* for some aspects of reality; they *may or may not correspond* to reality, which means that models can be misleading. The medieval model of human reproduction suggesting that babies develop from homunculi – fully developed bodies within the woman’s womb – leads to the absurd conclusion that the human race would become extinct after a finite number of generations (unless there is an infinite number of homunculi nested within each other). The model of a flat, disk-shaped earth may have prevented many navigators from exploring the oceans beyond the nearby coastal regions because they were afraid of “falling off” at the edge of the earth disk. The model of the falling profit rate in Marx’s economic theory predicted the self-destruction of capitalism, since the progress of productivity is reflected in a decreasing number of labor hours relative to the capital. According to this theory, labor is the only factor that adds plus-value to the products.

Schumpeter agreed on Marx's prediction, but based his theory on a very different model: Capitalism will produce less and less innovative entrepreneurs who create profits! The last two examples show that very different sophisticated models can sometimes lead to the same conclusions.

In neurology, artificial neural networks, consisting of a connection weight matrix, could be used as models for the functioning of the brain. Of course, such a model abstracts from all aspects except the connectivity that takes place within the brain. However, some neurologists believe that only 5%(!) of information passes through the synapses. If this turned out to be true, artificial neural nets would indeed be inappropriate models for the functioning of the brain.

One can see from these examples that models are ubiquitous and omnipresent in our lives. "The whole history of man, even in his most non-scientific activities, shows that he is essentially a model-building animal" [55] (see also [60]). We live with "good" and "bad", with "correct" and "incorrect" models. They govern our behavior, our beliefs, and our understanding of the world around us. Essentially, we see the world by means of the models we have in mind. The value of a model can be measured by the degree to which it enables us to answer questions, to solve problems, and to make correct predictions. Better models allow us to make better decisions, and better decisions lead us to better adaptation and survival – the ultimate "goal" of every being.

1.1.2 The Advent of the Computer

This paper is not about models in general, their variety of functions and characteristics. It is about a special class thereof: mathematical models. Mathematics has always played a fundamental role in representing and formulating our knowledge. As sciences advance they become increasingly mathematical. This tendency can be observed in all scientific areas irrespective of whether they are application- or theory-oriented. But it was not until last century that formal models were used in a systematic way to solve practical problems. Many problems were formulated mathematically long ago, of course. But often they failed to be solved because of the amount of calculation involved. The analysis of the problem – from a practical point of view at least – was usually limited to considering small and simple instances only.

The computer has radically changed this. Since a computer can calculate extremely rapidly, we are spurred on to cast problems in a form which they can manipulate and solve. This has led to a continuous and accelerated pressure to formalize our problems. The rapid and still ongoing development of computer technologies, the emergence of powerful user environment software for geometric modeling and other visualizations, and the development of numerical and algebraic manipulation on computers are the main factors in making

modeling – and especially mathematical modeling – an accessible tool not only for the sciences but for industry and commerce as well.

Of course, this does not mean that by using the computer we can solve every problem – the computer has only pushed the limit between practically solvable and practically unsolvable ones a little bit further. The bulk of practical problems which we still cannot, and probably never will be able, to solve efficiently, even by using the most powerful parallel machine, is overwhelming. Whether quantum-computer will change this situation, will be seen in the future. Nevertheless, one can say that many of the models solved routinely today would not be thinkable without the computer. Almost all of the manipulation techniques in mathematics, currently taught in high-schools and universities, can now be executed both more quickly and more accurately on even cheap machines – this is true not only for arithmetic calculations, but also for algebraic manipulations, statistics and graphics. It is fairly clear that all of these manipulations are already standard tools on every desktop machine. Sixth years ago, the hand calculator replaced the slide rule and the logarithm tables, now the computer replaces most of those mathematical manipulations which we learnt in high-school and even at university.

1.1.3 New Scientific Branches Emerge

Some research communities such as *operations research* (OR) owe their very existence to the development of the computer. Their history is intrinsically linked to the development of methods applicable on a computer. The driving force behind this development in the late 1940s was the Air Force and their Project SCOOP. Numerical methods for linear programming (LP) were stimulated by two problems they had to solve: one was a diet problem: Its objective is to find the minimum cost of providing the daily requirement of nine nutrients for a soldier from a selection of seventy-seven different foods. Initially, the calculations were carried out by five computers¹ in 21 days using electromechanical desk calculators. The simplex method for linear programming (LP), discovered and developed by Dantzig in 1947 , was and still is one of the greatest successes in OR. Together with good presolve techniques we can solve today almost any LP problems with millions of variables and constraints in a fraction of time.

Artificial intelligence and deep learning are also thoroughly dependent on the progress in computer science. Initially, many outstanding researchers in this domain believed that it would only be a question of decades before the intelligence of a human being could be matched by machines. Even Turing was confident that it would be possible for a machine to pass the Turing Test

¹ Human calculators carrying out extended reckoning were called “computers” until the end of the 1940’s. Many research laboratories utilized often poorly paid human calculators – most of them were women.

by the end of the century. Some scientists believe that we are not far from that point. Even though most problems in AI turned out to be algorithmically hard, since they are closely related to combinatorial problems. This led to an intensive research of heuristics and “soft” procedures – methods we humans use daily to solve complex problems. The combination of these methods and the computer’s extraordinary speed in symbolic manipulation produces a powerful means to implement complex problems in AI.

Other scientific communities had already developed highly efficient procedures for solving sophisticated numerical problems before the first computer was built. This is especially true in physics and engineering. For example, Eugène Delaunay (1816-1872) made a heroic effort to calculate the moon’s orbit. He dedicated 20 years to this pursuit, starting in 1847. During the first ten years he carried out the hand calculation by expressing the differential system as a lengthy algebraic expression in power series, then during the second ten years he checked the calculations. His work made it possible to predict the moon’s position at any given time with greater precision than ever before, but it still failed to match the accuracy of observational data from ancient Greece. A hundred year later, in 1970, André Deprit, Jacques Henrard and Arnold Rom revised Delaunay’s calculation using a computer-algebra system. It took 20 hours of computer time to duplicate Delaunay’s effort. Surprisingly, they found only 3 minor errors in his entire work.

Many numerical algorithms, such as the Runge-Kutta algorithm and the Fast Fourier Transform (FFT), were already known – the latter even by Gauss – before the invention of the computer and they were much used by human “computers”. But for many problems these efforts were hopeless, for the simple reason that the human computer was too slow to execute the simple but lengthy arithmetics.

An interesting illustration of this point is the origin of numerical meteorology. Prior to World War II, weather forecasting was more of an art, depending on subjective judgement, than a science. Although, in 1904, Vilhelm Bjerknes had already elaborated a system of 6 nonlinear partial differential equations, based on hydro- and thermodynamic laws, to describe the behavior of the atmosphere, he recognized that it would take at least three months to calculate three hours of weather. He hoped that methods would be found to speed up this calculation. The state of the art did not fundamentally change until 1949 when a team of meteorologists – encouraged by John von Neumann, who regarded their work as a crucial test of the usefulness of computers – fed the ENIAC with a model and got a 24-hour “forecast”, after 36 hours of calculations, which turned out to be surprisingly good. Four years later, the Joint Numerical Weather Prediction Unit (JNWP) was officially established; they bought the most powerful computer available at that time, the IBM 701, to calculate their weather predictions. Since then, many more complex models have been introduced. The computer has transformed meteorology into a mathematical science.

In still other scientific communities, until very recently, mathematical modeling was not even a topic or was used in a purely academic manner. Economics is a good example of this. Economists produced many nice models without practical implications. Sometimes, such models have even been developed just to give more credence to policy proposals. But mathematical models are not more credible simply because they are expressed in a mathematical way. On the other hand, important theoretical frameworks – such as game theory going back to the late twenties when John von Neumann published his first article on this topic – have been developed. Realistic n-person games of this theory cannot be solved analytically. They need to be simulated on computers. So the attitude towards these formal methods is also gradually changing in these other sciences as well. Today, no portfolio manager works without optimizing software. Branches such as evolutionary biology which have been more philosophical, are also increasingly penetrated by mathematical models and methods.

1.1.4 The Consequences

We should not underestimate the significance of the development of computers for mathematical modeling. Their capacity to solve mathematical problems has already changed the way in which we deal with and teach applied mathematics. The relative importance of skills for arithmetic and symbolic manipulation will further decrease. It will be more important to *understand* the concept of a derivative than to calculate it. And we will need more persons qualified to *translate real-life problems into formal language*, and what activity is more rewarding and intellectually more challenging in applied mathematics than just *that* – namely modeling? While relatively fewer mathematicians are needed to solve a system of differential equations or to manipulate certain mathematical objects – these are tasks better done by computers – more and more persons are needed who are skilled and expert in modeling – *in capturing the essence, the pattern and the structure of a problem and to formulate it in a precise way*.

This development is by no means confined to science. Since World War II, a growing interest has been shown in formulating mathematical models representing physical processes. These kinds of models are also beginning to pervade our industrial and economical processes. In several key industries, such as chip production or flight traffic, optimizing software is an integral part of their daily activities. In many other industrial sectors, companies are beginning to formalize their operations: Planning the workforce, sport scheduling, production planning, route planning for transportation, just to mention a few. Mathematical models are beginning to be an essential part of our highly developed society.

An important condition for the widespread use of mathematical modeling tools is a change in the mathematical curriculum in school: A greater part of

the manipulation of mathematical structures should be left to the machine, but more has to be learnt about how to recognize a mathematical structure when analyzing a particular problem. It should be an important goal in applied mathematics to foster creative attitudes towards solving problem and to encourage the students' acquisition and understanding of mathematical concepts rather than drumming purely mechanical calculation into their heads. Only in this way can the student be prepared for practical applications and modeling. **Science is modeling!** The ability to solve problems is modeling! "The great game of science is modeling the real world." (Hestenes, 1992, p. 732) [52].

But how can modeling be learnt? Problems, in practice, do not come neatly packaged and expressed in mathematical notation; they turn up in messy, confused ways, often expressed, if at all, in somebody else's terminology. Therefore, a modeler needs to learn a number of skills. She must have a good grasp of the system or the situation which she is trying to model; she has to choose the appropriate mathematical methods and tools to represent the problem formally; she must use software tools to formulate and solve the models; and finally, she should be able to communicate the solutions and the results to an audience, who is not necessarily skilled in mathematics.

It is often said that modeling skills can only be acquired in a process of learning-by-doing; like learning to ride a bike can only be achieved by getting on the saddle. It is true that the case study approach is most helpful, and many university courses in (mathematical) modeling use this approach. But it is also true – once some basic skills have been acquired – that theoretical knowledge about the mechanics of bicycles can deepen our understanding and enlarge our faculty to ride it. This is even more important in modeling industrial processes. It is not enough to exercise these skills, one should also acquire methodologies and the theoretical background to modeling. In applied mathematics, more time than is currently spent, should be given to the study of discovery, expression and formulation of the problem, initially in non-mathematical terms.

So, the novice needs first to be an observer and then, very quickly, a do-er. Modeling is not learnt only by watching others build models, but also by being actively and personally involved in the modeling process.

1.2. What is a Model?

In this section, a short introduction of the informal concept of model and related concepts is given. Then a precise definition of mathematical model is exposed and short examples explain the concept.

1.2.1 Modeling – an Informal Definition

The term model has a variety of meanings and is used for many different purposes. We use *modeling clay* to form small replica of physical objects; children – and sometimes also adults – play with a *model railway* or model aeroplane; architects build (scale) *model houses* or (real-size) model apartments in order to show them to new clients; some people work as *photo models*, others take someone for a model, many would like to have a *model friend*. Models can be much smaller than the original (an orrery, a mechanical model of the solar system) or much bigger (Rutherford's model of the atom). A diorama is a three-dimensional representation showing lifelike models of people, animals, or plants. A cutaway model shows its prototype as it would appear if part of the exterior had been sliced away in order to show the inner structure. A sectional model is made in such a way that it can be taken apart in layers or sections, each layer or section revealing new features (e.g. the human body). Working models have moving parts, like their prototypes. Such models are very useful for teaching human anatomy or mechanics.

The ancient Egyptians put small ships into the graves of the deceased to enable them to cross the Nile. In 1679, Colbert ordered the superintendents of all royal naval yards to build an exact model of each ship. The purpose was to have a set of models that would serve as precise standards for any ships built in the future. (Today, the models are exposed in the Musée de la Marine in Paris.) Until recently, a new aeroplane was first planned on paper. The next step was to build a small model that was placed in a wind-tunnel to test its aerodynamics. Nowadays, aeroplanes are designed on computers, and sophisticated simulation models are used to test different aspects of them.

The various meanings of *model* in the previous examples all have a common feature: a model is an imitation, a pattern, a type, a template, or an idealized object which *represents* the real physical or virtual object of interest. In the ship example, both the original and the model are physical objects. The purpose is always to have a copy of some original object, because it is impossible, or too costly, to work with the original itself. Of course, the copy is not perfect in the sense that it reproduces all aspects of the object. Only some of the particularities are duplicated. So an orrery is useless for the study of life on Mars. Sectional models of the human body cannot be used to calculate the body's heat production. Colbert's ship models were built so accurately that they have supplied invaluable data for historical research, this was not their original purpose. Apparently, the use made of these ship models has changed over time.

Besides these physical models, there are also *mental ones*. These are intuitive models which exist only in our minds. They are usually fuzzy, imprecise, and often difficult to communicate.

Other models are in the form of *drawings or sketches*,

abstracting away many details. Architects do not generally construct scaled-down models. Instead, they draw up exact plans and different two-dimensional projections of the future house. Geographers use topographical models and accurate maps to chart terrain.

Cave drawings are sketches of real animals; they inspired Picasso to draw his famous torro (see Figure 1.1). Picasso's idea was quite ingenious: how to draw a bull with the least number of lines? This exemplifies an essential characteristic of the notion of a model which is also valid for mathematical models, and for those in all other sciences: *simplicity, conciseness* and *aesthetics*. (More about these concepts see Chapter 2 of [76, 64].)

Certainly, what simplicity means depends on personal taste, standpoint, background, and mental habits. But the main idea behind simplicity and conciseness is clear enough: How to represent the object in such a way that we can "see" or derive its solution immediately? Our mind has a particular structure, evolved to recognize certain patterns. If a problem is represented in a way corresponding to such patterns, we can quite often immediately recognize its solution.

A illustrative example is the *intersection problem* (from preface written by Ian Stewart in [20]). Suppose you have to solve the following problem (Figure 1.2): Connect the small square A with F, B with D, and C with E inside the rectangle by lines in such a way that they do not intersect.

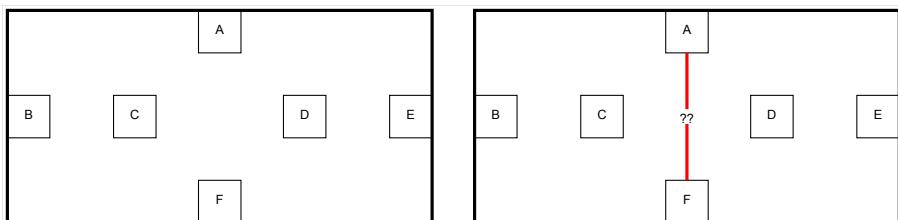


Figure 1.2: The Intersection Problem

At first glance, it seems that the problem is unsolvable, because if we connect A with F by a straight line, it partitions the rectangle into two parts, where B and C are in one part and D and E in the other. There is no way then,

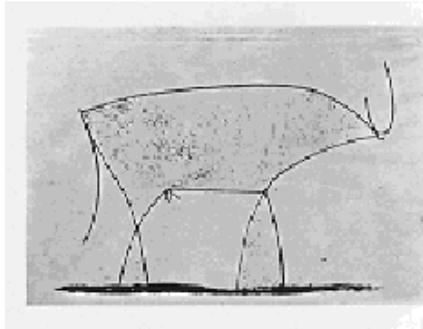


Figure 1.1: El Torro (P. Picasso, 1946)

to connect B with D or C with E, without intersecting the line A-F. Well, we were not asked to connect A and F with a *straight* line. So let us connect A with F by a curved line passing between say, B and C. But, then again we have the same problem: The rectangle is partitioned into two parts, B being isolated. The same problem arises when the line from A to F is drawn between D and E. But the problem can be presented a little bit differently: Imagine that the rectangle surface is spanned by a thin rubber. So pick D and C (by two fingers) and rotate the rubber with the two fingers by 180 degrees around the center of the rectangle in a continuous way such that the places of C and D are interchanged (Figure 1.3).

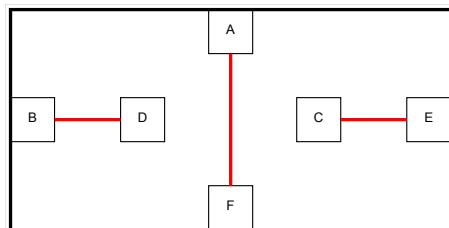


Figure 1.3: Topological Deformation

Now the problem is easily solvable. Connect the squares as prescribed. After this, return the rubber to the initial state again (Figure 1.4).

(A completely different approach to solve this problem can be found in my [My Books](#) – which is a good (advanced) exercise in mathematical modeling, see also the implemented model at [intersec](#).)

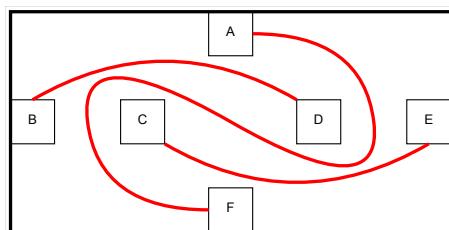


Figure 1.4: Solution to the Intersection Problem

Let's summarize:

- The model is the result of a *mapping* (or a transformation) of the real or virtual object to be considered.
- the model is an *abstraction*, i.e. only some aspects of the objects of interest are mapped, all others are ignored (*abstrahere* = leaving out).

- the aspects that enter the model are of *special interest* for an intelligent actor (the model-builder and the model-user): The model is used for a certain *purpose*.
- a model has some simple, concise (and aesthetic) *structure* that captures the focused aspects of the object.

Normally, the model is a simplification, but it must be rich enough to capture the problem at hand. Hence, it is something relative, there is no single absolute representation of the problem at hand. It is not only a model *of something*, but also *for someone* and *for some purpose*.

A good example is modern physics: Neglecting the Planck constant (that is: $\hbar = 0$), the gravitational constant ($G = 0$) and the speed of light ($c = \infty$), physics reduces to *classical mechanics*. Considering the gravitational constant ($G = 6.6741 \times 10^{11} [\text{m}^3 / (\text{kg} \cdot \text{s})]$) in addition, we must use *Newtonian mechanics*; Considering the speed of light ($c = 2.9979 \times 10^8 [\text{m/s}]$ instead, we must apply *special relativity*, and so on (see [35]). It is not true, that the models of classical mechanics are learnt, because it is necessary for the “more advanced” theory as Quantum Field Theory. It is useful and applicable *for its own*, namely when the speed is small, when quantum effects are negligible, and when gravitation is not important – that is in our every day life. Classical mechanics has plenty of applications on its own. Each model in physics has its own eligibility and purpose, it will do to know its limitation.

Hence, we saw that a problem can be approached with different models. The contrary is also true: A (formal) model might have various *interpretations*. The intersection problem was built of a rectangle with 6 locations in it (A, B, C, D, E, and F) and connecting lines. So it is basically a *geometric interpretation*. Another interpretation is as following: Replace “rectangle” with “square electric board”, “locations” with “endpoint of wires”, and “lines” with “wires”. Now we have the same model for a different problem: connect the endpoints of the wires on the 2-dimensional board in such a way that they do not cross. Another interpretation is: a oval closed room (with only one entry-point) has corresponding (curved) walls (from a point A to F, B to D and C to E, see the red lines in Figure 1.3) such that the whole room can be visited. Still another interpretation is: in an area we have six airports arranged in a similar way as the 6 locations in our original problem. Planify flight paths in such a way that the aeroplanes never cross each other. (We suppose that only the corresponding airports are linked, of course.)

An different question is whether the intersection problem is the right model for the flight-colliding problem. This could probably be solved in a more satisfactory manner by introducing different flying altitudes for the different connections. The wire-connection problem could probably be solved by insulating the wires and letting them intersect.

1.2.2 Mathematical Models

This power of abstraction, this ability to leave out some or many details about a concrete problem in order to use the “distillate” for a multitude of similar problems, is one of the most striking characteristics of our mind. A concrete, “real physical” problem need not even be there: We can make some useful considerations without a specific application in mind, as we have seen with the intersection problem. Mathematics and logic are essential tools in this process of abstraction. They are powerful languages to express and represent real phenomena. We could even define mathematics as the *science of finding and representing patterns and structures* of concrete or abstract problems. They are these patterns that are expressed in a mathematical model.

So, what is a mathematical model? Informally, it is a list of *alternatives* fulfilling certain *criteria* from which we choose one, often the “best”. Consider buying a house: we have a number of houses from which we choose one.

Formally, a mathematical model consists of symbols such as *operators* (arithmetic, Boolean, functions), *data* (the known entities), *variables* (the unknown entities), and *constraints* (the conditions). On a most basic level, we can define a mathematical model as a n -dimensional state space (or simply as a set) fulfilling certain properties or conditions. Consider the problem of buying a house: the “state space” are all houses on the planet (or the houses in a certain region), the “variable” is the unknown house to choose, the “data” is our budget, etc. The “conditions or constraints” is our taste: “it must have at least 5 windows and 2 balconies”, (and here is a Boolean operator).

In mathematics, we use symbols for these concepts:

$$Y = \{x \in X \mid R(x)\}$$

Y is called the mathematical model (which is a **set**), x is a vector of variables, X is a vector space of dimension n (often \mathbb{R}^n or \mathbb{N}^n , but not necessarily), R are the constraints, that is, a function which maps every element x in X into true or false: $R : X \rightarrow \{\text{true}, \text{false}\}$. By *solving a model* we mean the find one or all $x^o \in X$ such that $R(x^o)$ is true. The problem is said *infeasible* if no such x^o exists, in other words if the set Y is empty, otherwise it is said to be *feasible*.

Consider our house-buying problem: X is the set of all houses: house1, house2, house3, ..., etc. X is one-dimensional in this case, such a list of all houses. x is one unknown house out of all houses. $R(x)$ is the required condition of each house x , it is true or false for every house: house1=true [$R(x) = \text{true}$, $x = \text{house1}$] (yes, it has at least 5 windows and 2 balconies), $R(\text{house2})=\text{false}$ ((no, it is not true that it has at least 5 windows and 2 balconies)), $R(\text{house3})=\text{false}$, etc. Y is the list of all houses that have a true-property: $Y = \{\text{house1}, \dots\}$, the list of *eligible* houses.

Let us give several examples (please go carefully through these examples to understand the concepts well).

Example 1:

$$x \in \mathbb{N}^+, R(x) \stackrel{\text{def}}{=} (x = 1)$$

The state space consists of all positive numbers (\mathbb{N}^+). x is a singleton variable which must be a positive integer, and its value is 1 by definition of R , that is, we choose exactly one number which value is true, namely 1. Hence $Y = \{1\}$, the model is feasible and has exactly one solution, namely $x = 1$. (It is as if we have only one house with at least 5 windows and 2 balconies, that the unique one we choose!)

Example 2:²

$$x \in \mathbb{N}^+, R(x) \stackrel{\text{def}}{=} (x = 1 \wedge x = 2)$$

x is a singleton variable which must be a positive integer, and its value is 1 and at the same time 2 by definition of R . Hence $Y = \{\}$ (the set Y is empty), the model is infeasible, because there is no x which value is 1 *and* 2 at the same time. This is a contradiction. (It is as if no house fulfills our requirements.)

Example 3:

$$x \in \mathbb{R}^2, R(x) \stackrel{\text{def}}{=} (|x_1| \leq 1 \wedge |x_2| \leq 1)$$

The state space is 2-dimensional (every tuple of two real numbers is a possible choice). x is a 2-dimensional vector variable ($x = (x_1, x_2)$) with the components x_1 and x_2 , they must be real numbers each, and its absolute values must be smaller or equal 1 by definition of R . Hence, the model is given by $Y = \{(x_1, x_2) \mid -1 \leq x_1, x_2 \leq 1\}$, the model is feasible, and has an infinity number of solutions: every point in a 2-dimensional Euclidean space included in the 2×2 unit square is a solution: so: $(-0.5, 0.3), (0, 0), (0.1, 0.2), (0.4, 1)$, etc. are solutions. (Keeping our analogy with the choosing a house problem: We choose a house (x_1) together with a garage (x_2 .)

Example 4:

² Note that the symbol \wedge is the Boolean and operator. This means that both $x = 1$ and $x = 2$ must be true to make the whole expression true.

$$x \in \mathbb{R}^2, R(x) \stackrel{\text{def}}{=} (x_2 = x_1 + 2 \wedge x_2 = x_1^2 - 1)$$

The state space is again 2-dimensional. x is a 2-dimensional vector variable $x = (x_1, x_2)$ with the components x_1 and x_2 that must be real numbers, and they fulfill the two conditions defined by R . The solutions are:

$$\begin{aligned} Y &= \left\{ \left(\frac{1+\sqrt{13}}{2}, \frac{5+\sqrt{13}}{2} \right), \left(\frac{1-\sqrt{13}}{2}, \frac{5-\sqrt{13}}{2} \right) \right\} \\ &= \{(2.3, 4.3), (-1.3, 0.7)\} \end{aligned}$$

The model is feasible, and has exactly two solutions. To find the solutions, we need to solve a quadratic equation. Geometrically, the problem can be drawn in a 2-dimensional space (see Figure 1.5), the two solutions are where the line meets the parable.

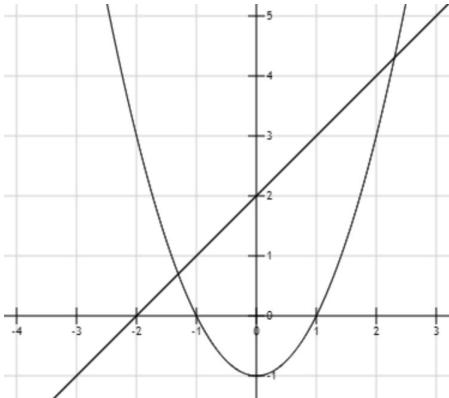


Figure 1.5: Geometric Solution

Some mathematical models are easy to solve (the examples so far), others are very difficult. For example the last conjecture of Fermat is a difficult model:

Example 5:

$$x = (a, b, c, n) \in \mathbb{N}^4, R(x) \stackrel{\text{def}}{=} (a^n + b^n = c^n \wedge n > 2)$$

This model is infeasible, that is, it has no solution. However, this 3-hundred year old conjecture has been proven by the mathematician A. Wiles only at 1997, after 9 years of intensive research.

Interestingly, a generalization by Euler, namely:

$$x = (a, b, c, d, n) \in \mathbb{N}^5, R(x) \stackrel{\text{def}}{=} (a^n + b^n + c^n = d^n)$$

has solutions. But it was not until 1960 to find one (others were found later on³):

$$95899^4 + 27519^4 + 27519^4 + 414560^4 = 422481^4$$

Even worse, it is easy to formulate models that cannot be solved because the computer takes far too much time to find a solution, for other models one can prove that they are unsolvable by any method.

Optimisation Models

A variant of a mathematical model is the *optimization problem*. In this case, we are not interested in any or all feasible solution, but only those solutions that maximize (or minimize) a function $f(x)$ (in analogy of the house choosing problem: we want the “best” house only).⁴

$$Y = \{\max f(x) \mid x \in X \wedge R(x)\}$$

Example 6: A trivial example is maximizing a parable in the whole real 1-dimensional space:

$$x \in \mathbb{R}, f(x) = -(x - 1)^2, R(x) \stackrel{\text{def}}{=} (\text{true})$$

The solution is $x = 1$, because the parable has a maximum at $x = 1$ where $f(x) = 0$ (for all other x the function value $f(x)$ is smaller than 0 (try it out!).

Example 7:

Now let us construct a real but easy problem. Answer the following question “What is the height and the width of a cylindric aluminium can that contains

³ 1966: $27^5 + 84^5 + 110^5 + 135^5 = 155^5$ or 1988: $2682440^4 + 15365629^4 + 18796760^4 = 20615673^4$

⁴ A optimization problem is commonly noted as

$$\begin{array}{ll} \min & f(x) \\ \text{subject to} & R(x) \\ & x \in X \end{array}$$

I introduce a different notation to show clearly that a mathematical model is a *set* and this definition is more general:

$$Y = \{\min f(x) \mid x \in X \wedge R(x)\}$$

a volume of 1 liter (dm^3), such that the quantity of material to build the can is as small as possible?"

Supposing that the material has identical thickness on its surface, we may substitute "quantity of material" by "surface of the cylinder". Given the height h and the diameter $2r$ of a cylinder, its volume is given as follows (which must be 1):

$$V = \pi h r^2 = 1$$

The surface O of the cylinder is then as follows (two circles and a cylinder coat):

$$O = 2\pi r^2 + 2\pi r h$$

We need to minimize the surface O (which we suppose to be proportional to the material usage, as already mentioned). Hence, the model is as follows:

$$x = (r, h) \in \mathbb{R}^{2+}, \quad \min f(x) \stackrel{\text{def}}{=} 2\pi r^2 + 2\pi r h, \quad R(x) \stackrel{\text{def}}{=} (\pi h r^2 = 1)$$

One can also simplify the model by eliminating the variable h . We then get the model:

$$O = 2\pi r^2 + \frac{2}{r}$$

where O must be minimal.

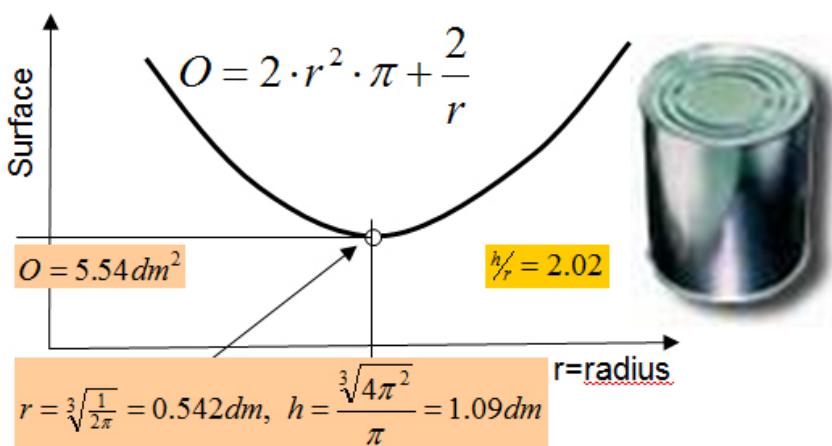


Figure 1.6: Solution of the can problem

This is a minimization problem with one variable and no constraint. The solution is given in Figure 1.6. The minimal usage of material to construct a cylindric can that contains a volume of 1 liter has a radius of 5.42cm and a height of 10.9cm, and the surface is 554cm². Any other size has a larger surface, check it!

Another way to express the can problem – with two variables – is as follows:

$$\begin{array}{ll} \min & 2\pi r^2 + 2\pi rh \\ \text{subject to} & \pi hr^2 = 1 \\ & r, h \geq 0 \end{array}$$

Basically, every mathematical model is like our house choosing problem, only the size of the state space change and the requirements are expressed in a mathematical notation. At this point, I could introduce a really powerful concept to formulate *large and complex* mathematical models, that is, models with many variables and constraints. This concept is called *indexing*. Since it is a so important and powerful notation, I wrote a separate paper for that (see [67]). If you want to get a pro in mathematical modeling, these concepts are a must!

A historical note: The concepts of the mathematical model are historically recent ones. According to Tarski, “the invention of variables constitutes a turning point in the history of mathematics.” Greek mathematicians used them rarely and in an ad hoc way. Vieta (1540-1603) was the first who made use of variables on a systematic way. He introduced vowels to represent variables and consonants to represent known quantities (parameters). “Only at the end of the 19th century, however, when the notion of a quantifier had become firmly established, was the role of variables in scientific language and especially in the formulation of mathematical theorems fully recognized” [1] p.12.

1.2.3 Related Concepts

For practical purposes, it is useful to define briefly some other related terms. The creator of a model is called the *modeler*. The modeler needs to have a profound knowledge of the object to be modeled, as well as an extensive understanding of mathematical structures in order to match them to the problem at hand. To create a model is and will always remain a complex and involved activity, even if computer based modeling tools can support the modeler, it cannot replace the creative work a modeler does. Of course, the modeler can be a team rather than a single person, as is often the case in big projects. In

this case, the different activities such as data collection, setting up the model, solving the model, and verifying it, etc. can be distributed between the modellers.

Another concept is *modeling*. The term is difficult to define because it is used in so many different contexts. In art, it is used to express the process of working with plastic materials by hand to build up forms. In contrast to sculpturing, modeling allows corrections to be made on the forms. They can be manipulated, built and rebuilt until its creator feels happy with the result. In sculpturing, the objective cannot be formed, it must be cut out. Once finished, the form cannot be modified. Scientific activities are more like modeling than sculpturing, and therefore the common-sense meaning of modeling is used in various research communities.

In database technology, for instance, *data modeling* designates the design, validation, and implementation of databases by applying well-known methods and approaches such as the entity-relationship (ER) approach and the object-role modeling (ORM) method. From a practical point of view, data modeling could be considered as part of mathematical modeling. It is a set of tasks which consists of collecting, verifying, and ordering data.

Hence: *Modeling is an iterative process to conceptualize, to build, to validate, and to solve a model.* Often modeling is more important than the end-product, the model, to understand a problem. The creative process of getting though the modeling process gives invaluable insights in the structure of a particular problem.

(The concept of modeling process is so important that I have written a separate paper (see [66] (How to model?).)

Two further important notions in mathematical modeling are that of *model structure* and *model instance*. A model structure is a model where all data is captured by symbolic entities: the parameters.⁵ In fact, it represents a whole class of models. A *model instance*, on the other hand, does not contain any parameters; these are replaced by concrete data. As an example, take a look at the following differential equation:

$$\frac{dx}{dt} = \frac{rx(1-x)}{k}$$

The equation contains two parameters r and k (and a variable x). Hence,

⁵ The symbols with known values are called *parameters* and the symbols with unknown values are called *variables* – as already used above. Since these two terms clashes with well known concepts in programming languages, the convention is used to call the parameters in functions and procedures headings *formal parameters*. Using these parameters in a function or procedure call will be called *actual parameters*. The term variable used in programming languages, which is a name for a memory location, will be called *memory variable*.

it defines an entire class of sigmoid functions and can, therefore, be used to model the growth of a population with limited resources. For a *concrete* growth model, however, the two parameters r and k must be replaced by numerical data. Additionally, an initial population x_0 must be given to make the equation numerically solvable.

Large models (models with many 1000 of variables and constraints) may also contain a large number of data that must first be collected, analyzed and verified. This may be a laborious task in itself, but that is another whole story.

Finally, a *modeling language* is a computer executable notation that represents a mathematical model augmented with some high level procedural statements to proceed the data of the model. There is an important difference between programming and modeling languages. Whilst the former encodes an algorithm and can easily be translated to a machine code to be executed, the later *represents* a mathematical model. It does normally not have enough information in itself to generate a solution code. So, why should modeling languages be useful? First of all, such a computer executable notation encodes the mathematical model in a concise manner, and it can be read by a computer. The model structure can be analyzed and translated to other forms. Secondly, for many important model classes the notation can be processed and handed over to sophisticated algorithms which solve the problem without the intervention of a user. Modeling languages have many further benefits, for example, automatic model documentation, generating activity-/constraint graph or similar graphical representations, etc. (There is another paper that explains my concept of a modeling language (see [63]).

CHAPTER 2

HOW TO MODEL?

“Begin to weave and God will give the thread.”

This paper has three parts. It gives first a short introduction on the modeling process, explains very briefly its stages in order to learn how to model. In a second part a number of small problems with solutions in a third part are exposed by which the reader can exercise the skills of modeling. To be useful, the reader should really try hard to solve these problems and look up the solution only after trying. It is the only way to acquire the skill of modeling. Three problems are from [43] which is an excellent book with more than 150 puzzles. Many Internet pages and books are available with puzzles, one of the oldest is [26]. Martin Gardner has also published a number of mathematical puzzles in book form. The focus here is mainly on mathematical models, hence the third part of the problems have also mathematical formulations if appropriate. Many modeling examples of puzzles can also be found in my puzzle and other book: See [My Modeling Books](#).

2.1. Introduction

Modeling is an art! But modeling is also a skill that can be learnt. Every problem to be solved is new, therefore, there is no unique recipe to model and solve it, but there are guidelines. The most important ability, however, is *to practise, to practise and to practise*. Looking at the problems, try to solve them, analyzing and reproducing how other persons solved them. In this way, the learner gains experience and security. Also you should learn the theoretical concepts going along the modeling process. For example, if the concept of “graph” is used, you must learn the basic concepts of graph theory, if modeling a “linear mathematical model”, the basic concepts of linear algebra must be learnt.

Basically, The process of modeling goes in stages:

Recognition – formulation – solution – validation

Recognition is the stage where the problem is identified and the importance of the issue is perceived; *formulation* is the most difficult step: the model is set up and a mathematical structure is established; *solution* is the stage where an algorithm is chosen or conceived to find a solution; finally, the model must be *interpreted* and *validated* in the light of the original problem: Is it consistent? Does it correspond to reality? etc. Modeling, however, is essentially an iterative process where the different stages must be repeated. The reasons are manifold: A first attempt often does not produce a satisfactory model to the original problem. It is often too crude or simply inaccurate or even wrong in representing the original problem. When inaccurate, the model has to be *modified*; when too crude, it must be *refined*; when too inappropriate the model has to be *rejected*. This modify-refine-reject process is very important and it is the rule and not the exception. Keep this in mind! Every step

is new and a creative process that can go in a wrong direction, so *perseverance* is another prerequisite a modeler must acquire to be successful. Going from stage to stage also means discovering – uncovering – new structures and connections that were previously unknown to the model builder. This is unavoidable as the modeling process is not a mechanical activity, where the intermediate and final goals are known in advance. The process is fundamentally non-predictable. The evolution of a model is not unlike that of paradigms described by the history of science [78]. “Normal” periods of successive small improvements and corrections are followed by “periods of crisis”, during which the basic statements are radically questioned and eventually replaced. For these reasons, models should always be regarded as tentative and subject to continued evaluation and validation. Remember it is easy to falsify, but not easy to verify a model! Let us have a closer look at the four stages.

2.1.1 Recognition and Specification of the Problem

First of all, it must be said that very little progress can be made in modeling without fully understanding the problem that has to be modeled. If the modeler is not familiar with the real problem, she will never be able to build a model of it. Traditionally, not much mathematics is involved at this stage. The problem is studied for an extended time, data is collected and empirical laws or guidelines are formulated before a systematic effort is made to provide a supporting model. The first mistake comes from an erroneous idea that the final and formalized model and its solution are more important than this “preliminary” stage, although a careful and accurate study of the problem is essential. The second mistake often arises because many modelers think that using *some* formalism is more important than using an appropriate notation that reflects the problem. Yet the most sophisticated formalism is useless if it does not mirror some significant features of the underlying problem. Often, the modelers only have access to a limited arsenal of modeling tools, and they apply what they have at hand. Appropriateness comes second, so the erroneous approach often goes!

Hence, the very beginning of the modeling process needs expertise and a good eye to filter out unimportant factors. As difficult, and sometimes as tedious as the modeling process may be, there is no way around it: It is impossible to specify a problem without *setting up an informal model*. Any specification is a model using some language or symbolism. The first formulation is mostly in natural language. Sketches or drawings usually accompany the formulation. At this stage, the objectives should be clearly stated. A plan on how to attack the problem has to be put forward. In textbooks on problem solving in applied mathematics, this first stage is assumed, but in facing a real problem, it is often not even clear what the problem consists of. Unless the modeler gets this right at the start, it is of little use to build a formalized

model.

Summary: Repeat the problem in your own words! Make sketches. Are there data? What is the goal? What are the conditions? Explain the problem to another person! Do not stop before you understand the problem entirely. Order the facts! Name the components! Do not hesitate to restart!

2.1.2 Formulation of the (Mathematical) Model

Mathematics is the science of searching for patterns and for structures. This is intellectually the most rewarding activity, it is undoubtedly creative. Translation a problem into mathematical notation uses mathematical knowledge and skills as well as problem expertise. But it also requires something more: the talent to recognize what is essential in a particular context and to discard the irrelevant, as well as an intuition as to which mathematical structure is best suited to the problem. This intuition cannot be learned as we learn how to solve mathematical equations. Like swimming, this skill can be acquired by observing others and then trying by oneself. Hence, it is difficult to provide an adequate formal description on the model formulation process.

Since the focus here is on *mathematical models* as defined in another paper (see [75]). On an abstract level, there is no doubt about what has to be done: defining the variables x and finding the constraints $R(x)$ for the model, as well as collecting the data and the parameters p . On a more concrete level, one can only give some guidelines, such as:

- Establish a clear statement of the objectives
- Recognize the key variables, group them (aggregate)
- Identify initial states and resources
- Draw flow diagrams
- Do not write long lists of features
- Simplify at each step
- Get started with mathematics as soon as possible
- Know when to stop.

Pólya, in his classical work [20] (see also [21]) about problem solving, tried to give heuristic strategies to attack mathematical problems – an essay that every person who does serious modeling should read. Pólya has summarized them as follows:

1. Understanding the problem: What is (are) the unknown(s)? What are the data? What are the conditions or constraints? Are they contradictory or are some of them redundant?

2. Devising a plan of attack: Do you know of a related problem? Can you restate the problem? Can you solve a part, a special case or a more general problem? Did you use all the data and all the essential notions of the problem?
3. Carrying out the plan: Solve the related problems. Check each step.
4. Looking back: Do you really believe the answer?

In the second part of this paper several examples are given.

2.1.3 Solving the Mathematical Model

Once the model formulation step has been completed, the mathematical model has to be solved, i.e. the model must be transformed using calculation in such a way as to eliminate the unknowns. We say that a model is solved if a numerical or symbolical expression containing only parameters and numbers is assigned to each variable so that all constraints are fulfilled.

The solution step can be very straightforward, as in most examples in the third part of this paper. Most of the time this is not the case. It is easy to formulate innocent looking models that are very difficult to solve. For certain model classes a well understood mathematical framework and numerical software exist to solve them. If not, the model must be reformulated to be tractable. Often it is not necessary to reformulate an intractable model; one can use *simulation techniques* or *heuristics* to find approximate – hopefully good – solutions to the model. Anyway, in all but trivial cases the solution cannot be carried out by hand, and one must develop computer programs to execute the lengthy calculations. But even if the mathematical theory of a problem is straightforward, the invention of an appropriate algorithm can require considerable ingenuity. Furthermore, it is not easy to write numerically stable and reliable software. Therefore, for many problem classes available software packages – called *solvers* – have been built by expert numerical analysts.

When using a solver, the model must be put into a form that can be read by the appropriate solver. This is a laborious and error-pruned task especially for large models with lots of data involved. Data must be read and filtered correctly from databases, the model structure must be reformulated, and the generated output must be a correct form for input into the solver. For this task, modeling tools such as (mathematical) *modeling languages* exist that can simplify greatly this task. (I have more to say about modeling languages in a separate paper: [70].)

2.1.4 Validation and Interpretation

Creating and building a model is a creative process where “everything goes”, but checking and validating the model is a completely different affair. Validation, in fact, is a continuous process and takes place at every stage not only in the final part. By validation we mean the process of checking the model against reality, and the strongest types of validation is *prediction*.¹ But validation is much more than inspecting the similarities or dissimilarities of the model against reality. It comprises a variety of investigations that help to make the model more or less *credible*. This consists of many questions that should be answered, such as: Is the model mathematically consistent? Is the data used by the model consistent? Is it commensurable? Does the model work as intended? Does it “predict” known solutions using historical data? Do experts in the field come to the same conclusions as the model? How does the model behave in extreme or in simplified cases? Is the model numerically stable? How sensitive is the model to small changes in input data? Let’s go through several elements:

1. **Logical Consistency:** Of course, a model must be free of contradiction, since from a inconsistent statement everything follows (“ex falso quodlibet”). This is trivial! However, in a practical model building process, inconsistencies arise in a natural way, mostly because of conflicting goals or hard constraints².
2. **Data type consistency:** This issue has been discussed in detail and over many years by the programming language community. Data type checking means to check the domain membership of a value before it is assigned to a storage object in computers.
3. **Unit Type Consistency:** Another issue is to check whether the data in expressions are commensurable. In a modeling context, dimensional checking is of utmost importance, since most quantities are measured in units (dollar, hour, meter, etc.). Experiences in teaching operations

¹ The discovery of the planet Neptune is an often quoted example. Since the discovery of the planet Uranus in 1781, perturbations had been noted in its orbit. An explanation within Newton’s gravitational model which could fit these observations was the presence of a previously unobserved planet. A telescope at Berlin was then pointed in the right direction at the right time, on the basis of Le Verrier’s extended calculations, and a new planet was found: Neptune. This example shows the predictive power of a good model. But history also shows how careful we should be about making general conclusions: Later, when irregularities in Mercury’s orbit were discovered, a similar explanation with a new planet, Vulcan, was proposed; but Vulcan was never discovered! These irregularities were only explained by Einstein’s general theory of relativity.

² When we say that a budget should by 1’000’000, then we normally do not mean that it must be *exactly* that amount, we mean “about 1’000’000”. Many model examples show how to handle these kind of situations (one is in my puzzle book, see [library](#))

research reveal that one of the most frequent errors made by students when modeling is inconsistency in the units of measurement. In physics and other scientific applications, as well as technical and commercial ones, using units of measure in calculating has a long tradition. It increases both the reliability and the readability of calculations.

4. **User defined data checking:** Do I have the right data? Are the data correct and consistent? Are the data complete? Often data checking is a huge task itself. But it is necessary, otherwise we get a garbage-in garbage-out model, that is, the model is only useful if the data make sense. The data should also be separated as much as possible from the model structure (see [75]). In this way, a data set can easily be replaced by another data set. Applying various data set (smaller data set or random sets, etc., in a prototyping preliminary stage) can largely help to find inconsistencies.
5. **Simplicity Considerations:** Can the model be simplified or reduced? The principle of Ockham's Razor also applies to modeling. To paraphrase Einstein: The model should be as simple as possible, but not simpler.
6. **Solvability Checking:** Formulate model that are difficult to solve is easy. One can try various solvers, but sometimes one can modify the model – for instance by adding trivial constraints – to help the solver to find a solution in shorter time. These techniques need a lot of experiences and cannot be exposed here (without going into the details: an small and simple example is the Pigeon hole model [pihole](#)).

The model creation process is a difficult undertaking. I barely touched a few highlights. What is important is to practice to gain experience and to study in parallel the branches in mathematics used in a particular context. What follows in a second part of this paper is a number of rather easy problems, that the reader can try to solve. First the problems are formulated, after a possible solution is presented with some remarks on modeling exposed in the previous sections.

2.1.5 The Modeling Life-Cycle

The model building process can also be viewed as cycle as depicted schematically in Figure 2.1.

The starting point is a “real world problem” (where “real world” is meant to be very general – it could be an abstract problem, like the 4-color problem of a planar graph, for example). The first step in modeling is to translate this problem into a precisely defined mathematical problem. Maybe a similar problem exists already and the modeler only need to “match” the real problem to the mathematical symbolism. Sometimes, however, a new theory or formalism

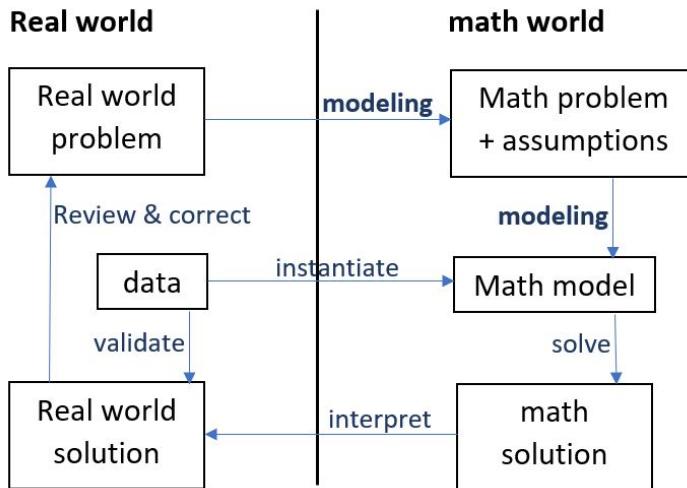


Figure 2.1: The Modeling-Life-Cycle

must be invented in order to capture the problem – Newton invented the infinitesimal calculus to explain the motion and dynamic change, such as the orbits of planets, the motion of fluids, etc.

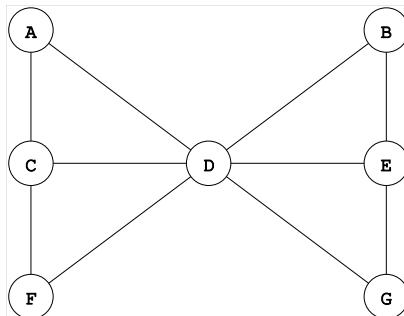
To solve a concrete “real” problem, the mathematical problem must be enriched with *assumptions* and *data*. For example, to find the shortest path from my home to the office, one needs to know what kind of vehicle to use, what are the distances, is there a traffic jam, etc. Hence, a second modeling step is to collect all the needed data, to add implicitly or explicitly all the relevant assumptions to build a concrete mathematical model, that is, to instantiate the model, which then can be solved. To solve a problem mean to transform the problem using mathematical operations and rules until the unknowns become known values. This solving step is mostly done by an algorithm – called *solver* – and executed on a computer.

Once the solution is known, it must be interpreted in the light of our “real” problem and validated by the data. Does the result make sense? Does it match the data? If not, we must review and correct our steps. Maybe one needs to modify the data, the assumptions, or change the whole approach all together. Sometimes completely new insights in the problem are discovered which can lead to a complete new “real problem” which appears to be more promising. An the whole modeling life cycle begins afresh.

2.2. The Problems

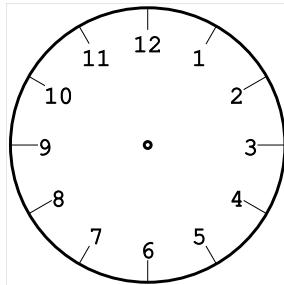
2.2.1 Problem 1: Seven Digits Puzzle

Use each digit from 1 to 7 exactly once, and place them into the circles of the Figure at the right side in such a way that the sum along each of the five straight lines is the same (for example, A-C-F is a straight line).



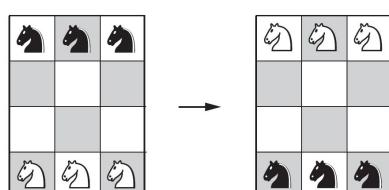
2.2.2 Problem 2: The Clock Puzzle

Divide the clock with a straight cut such that the sum of the numbers in both parts are equal?



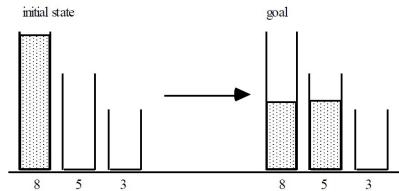
2.2.3 Problem 3: The 6-Knights Puzzle

Move the knights from a initial position (left part of the Figure) using the chess rules in a 3×4 board to a final position (right part). The puzzle is from [43].



2.2.4 Problem 4: The 3-jug Problem

There are three jugs with capacities of 8, 5, and 3 liters. Initially the 8-liter jug is full of water, whereas the others are empty. Find a sequence for pouring the water from one jug to another such that the end result is to have 4 liters in the 8-liter jug and the other 4 liters in the 5-liter jug. When pouring the water from a jug A into another jug B, either jug A must be emptied or B must be filled.



2.2.5 Problem 5: The Fake Coin Puzzle I

There are 8 identical-looking coins; one of these coins is counterfeit and is known to be lighter than the genuine coins. What is the minimum number of weighings needed to identify the fake coin with a two-pan balance scale without weights? The puzzle is from [43].

2.2.6 Problem 6: The Fake Coin Puzzle II

There are 12 coins identical in appearance; either all are genuine or exactly one of them is fake. It is unknown whether the fake coin is lighter or heavier than the genuine one. You have a two-pan balance scale without weights. The problem is to find whether all the coins are genuine and, if not, to find the fake coin and establish whether it is lighter or heavier than the genuine ones. Design an algorithm to solve the problem in the minimum number of weighings. The puzzle is from [43].

2.2.7 Problem 7: The Horse Race Puzzle

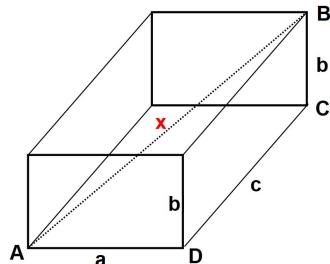
There are 25 horses. What is the minimum number of races needed to identify the 3 fastest horses? Up to 5 horses can race at a time, no watch is allowed.

2.2.8 Problem 8: Adding Numbers

Find the sum of all integers from 1 to 1000. Then find a formula for the sum of all integer from 1 up to an number $n > 1$ and proof it by induction.

2.2.9 Problem 9: 3d-Diagonal

Find the diagonal x of a rectangular parallelepiped of which the length c , the width a , and the height b are known.



2.2.10 Problem 10: Square Inside a Triangle

Inscribe a square in a given triangle. Two vertices of the square are on the base line of the triangle, and the other two vertices touch the two sides of the triangle. Are there more than one possibilities?

2.2.11 Problem 11: Birthday Paradox

Given a party of n persons, what is the probability $P(n)$ that at least 2 persons have the same birthday? For instance, if 30 persons are invited to a party, is it worthwhile to make a bet that two persons have the same birthday?

2.2.12 Problem 12: Length of Loch Ness Monster

If the length of the Loch Ness Monster is 20 meters and half of its length, how long is it?

2.2.13 Problem 13: Price of the Ball

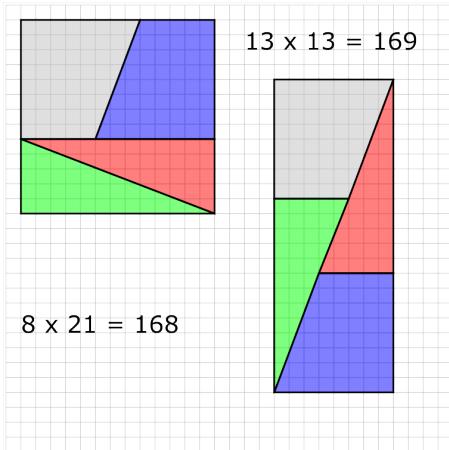
A golf club together with a ball costs 110 €. The price difference between a golf club and a ball is 10 €. What is the price of a ball?

2.2.14 Problem 14: Postman Problem

A postman distributes all letters in 45 minutes. If his assistant helps him, it takes only 20 minutes. How long would the assistant alone take?

2.2.15 Problem 15: One Unit Missing

The four forms in the 13×13 squares (left part) are apparently arranged in a rectangle of 8×21 . Where is the missing unit? The square has $13 \times 13 = 169$ unit cells, but the rectangle only has $8 \times 21 = 168$ unit cells. Also find the hidden structure in this problem and generalize the results.



2.2.16 Problem 16: How old am I?

My age is a third of my father's age, but in 10 years my father will only be two times as old as I will be. What is my age today?

2.2.17 Problem 17: 7-Digits Puzzle (redo)

This is the same as the Puzzle 1 above: Use each digit from 1 to 7 exactly once, and place them into the circles of the Figure in such a way that the sum along each of the five lines is the same. This time, create a mathematical model with variables and constraints to solve the problem.

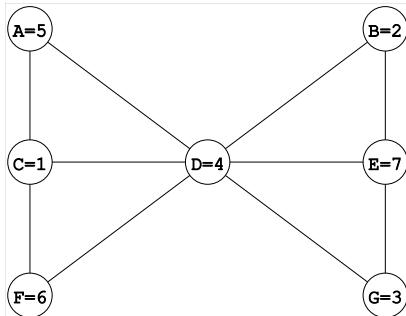
2.2.18 Problem 18: The Clock Puzzle (redo)

This is the same as Problem 2 above: Divide the clock with a straight cut into two parts such that the sum of the numbers in both parts are equal? This time, create a mathematical model with variables and constraints to solve the problem.

2.3. The Solutions

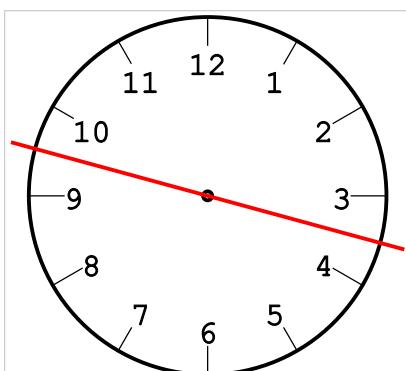
2.3.1 Problem 1: Seven Digits Puzzle

Loud thinking: “There are 7 numbers and 7 circles, this fact fits after all (If they were 8 circles and 7 numbers then this problem would not have a solution)! A straight line consists of three circles (hence three numbers). The figure shows some symmetry. The center circle seem to have a important position, because three lines cross it. Hmm... three number pairs, namely $1 + 7$, $2 + 6$, and $3 + 5$ sum up to 8 each. What if 4 is in the center? Of course!” The rest is easy: no many possibilities remain. When modeling repeat the facts over and over again in different words. Write them down.



2.3.2 Problem 2: The Clock Puzzle

Loud thinking: “If one side must have the same sum than the other, then it is unlikely that all large numbers are on one side and all small numbers are on the other side. A mixture is needed. What about putting the largest and the smallest together? $12 + 1 = 13$. The same sum is given by $11 + 2 = 13$. Well, all 12 numbers can be paired to give 13. There are 6 pairings. Put three of them on one side and the other three on the other side!”



To prove that this is the unique solution, we must show that there exists no other consecutive sequence of numbers on the clock face that sums to the half of all numbers. Let's choose two unknown numbers x and y such that $1 \leq x < y \leq 11$. The sum of all numbers on the face from $y + 1$ to x in clockwise direction is the sum from $y + 1$ to 12 plus the sum from 1 to x . This is: the sum of all numbers from 1 to 12 minus the sum of all numbers from 1 to y plus the sum of all numbers from 1 to x :³

³ Note that the sum of the numbers from 1 to n is given by $n(n + 1)/2$, (see Problem

$$\left(\frac{12 \cdot 13}{2} - \frac{y(y+1)}{2} \right) + \frac{x(x+1)}{2}$$

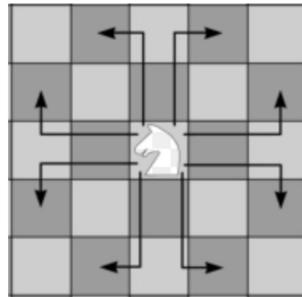
This quantity must be half of all numbers from 1 to 12, that is, it must be $\frac{12 \cdot 13}{4}$. Simplifying the expression leads to:

$$(y - x) \cdot (y + x + 1) = 6 \cdot 13$$

As $y - x < y + x + 1$, we conclude that $y + x + 1 = 13$ and $y - x = 6$. Resolving for x and y gives: $y = 9$ and $x = 3$. Hence the unique solution is a consecutive sequence from 10 to 3 in clockwise direction.

2.3.3 Problem 3: The 6-Knights Puzzle

This problem is a instructive example for the process of abstraction. In an initial step, the problem is reduced to its essential components. First we recall the rules of jumping for a single knight (see Figure at the right). Then the cells of the board are numbered from 1 to 12 to identify them.



The board and the knights do not matters, what only matters is from which number (cell) one can jump to another number: link these numbers with a straight line. The result is a well-known mathematical structure (a graph), consisting of lines connected to two locations (the numbers), see Figure 2.2 at the right. Note that from a knight board we have arrived at a very different structure (namely the numbers linked by straight lines), the problem has been reduced to the essential elements.

The positions of the numbers do not matter either, so we can “unfold” the graph and place the number wherever we want. However, the connecting lines must be preserved. Finally, we get a much simpler graph (see Figure 2.3 at the right), from which the solution can immediately be constructed: First the two knights in the middle are moved clockwise, till they have exchanged their places, then the outer knights are moved clockwise in the same way, each one move after the other.

Finally, the original problem can be reconstructed by executing the knights jumping in exactly that order. What is interesting about this puzzle – that is

⁷ below).

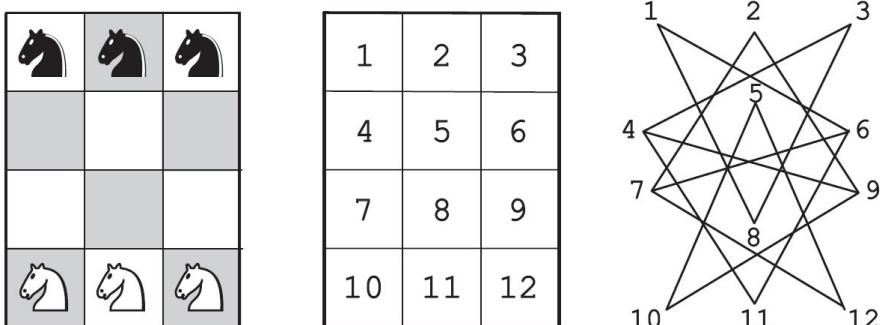


Figure 2.2: Transform the Problem to a Graph [43]

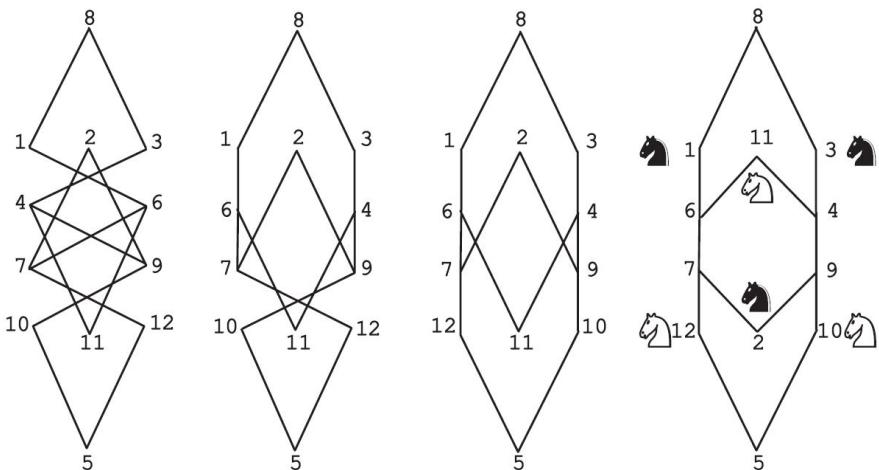


Figure 2.3: Simplifying a Graph [43]

important for most real modeling problems – is the fact that one can ask (and answer very easily) “what if questions”. In the solution above, for instance, knight 1 and knight 10 exchange their places. “What if” I want to exchange knight 1 with knight 13 instead? Is this possible? The reader would easily find a solution.

2.3.4 Problem 4: The 3-jug Problem

This is another example for using the concept of graph to model the problem. The problem can be formulated as a number of *states* and *transitions* between the states. A state is a particular filling of the jugs, for instance, a state is: “the 8-liter jug contains 8 liters of water, the 5-liter jug contains nothing, and

the 3-liter jug contains nothing.” Each state can be represented by a triple of numbers: (x, y, z) , where x is the content of the 8-liter jug, y is the content of the 5-liter jug, and z is the content of the 3-liter jug. So, $(8, 0, 0)$ is the example just used before. We want to reach the state $(4, 4, 0)$. The capacities of the jug can also be represented by a triple: $(8, 5, 3)$.

The first step is to enumerate all states. There are 216 potential states, namely, all states (x, y, z) with $0 \leq x \leq 8$, $0 \leq y \leq 5$, and $0 \leq z \leq 3$, which is $9 \cdot 6 \cdot 4 = 216$. However, only states with the condition $x + y + z = 8$ are allowed (no water should be added or removed). Furthermore, at least one jug must be full or empty – following the rules of pouring. This condition can be represented as a Boolean expression. $x = 0 \vee x = 8 \vee y = 0 \vee y = 5 \vee z = 0 \vee z = 3$.⁴ There are 16 remaining possible states, they are:

$$\{(0, 5, 3), (1, 4, 3), (1, 5, 2), (2, 3, 3), (2, 5, 1), (3, 2, 3), (3, 5, 0), (4, 1, 3), \\ (4, 4, 0), (5, 0, 3), (5, 3, 0), (6, 0, 2), (6, 2, 0), (7, 0, 1), (7, 1, 0), (8, 0, 0)\}$$

The basic operation is to pour water from one jug A to another jug B in a way that either A is emptied or B is filled. We are looking for the shortest sequence of operations that reaches the state $(4, 4, 0)$ starting with state $(8, 0, 0)$. Such a basic operation is called a *transition from one state to another*. You noticed? The “problem of the 3-jugs” has been transformed into a “problem of states and transitions between the states”. The final step is to position the states somewhere in the plane and link two states with an arrow, if a transition from the first to the second state is possible. The result is a (directed) graph, a graph with directed links. The problem now is reduced to find the shortest (direct) path⁵ in this graph from state $(8, 0, 0)$ to the state $(4, 4, 0)$. The resulting graph and the shortest path in red is given in Figure 2.4.

I guess it is clear how to interpret the solution: In the first step, take the 8-liter jug and fill the 5-liter jug, the second step is to take the 5-liter jug and to fill the 3-liter jug, the third step is to empty the 3-liter jug and pour it to the 8-liter jug, and so on, 7 steps are needed to get to the goal $(4, 40)$.

Interestingly, we not only know now how to solve this particular 3-jug problem but all kind of jug problems with other capacities and with a different number of jugs. This is a strong indication of a good model: if the model’s vocabulary stimulates others undiscovered aspects of the problem or guides the research to similar problems, then the model might be reasonably good. If, however, the vocabulary is “sticky” or “artificial” then the model is probably not very useful.

⁴ \vee is the Boolean “or” operator.

⁵ Finding the shortest path in a graph is a well-known problem in operations research.

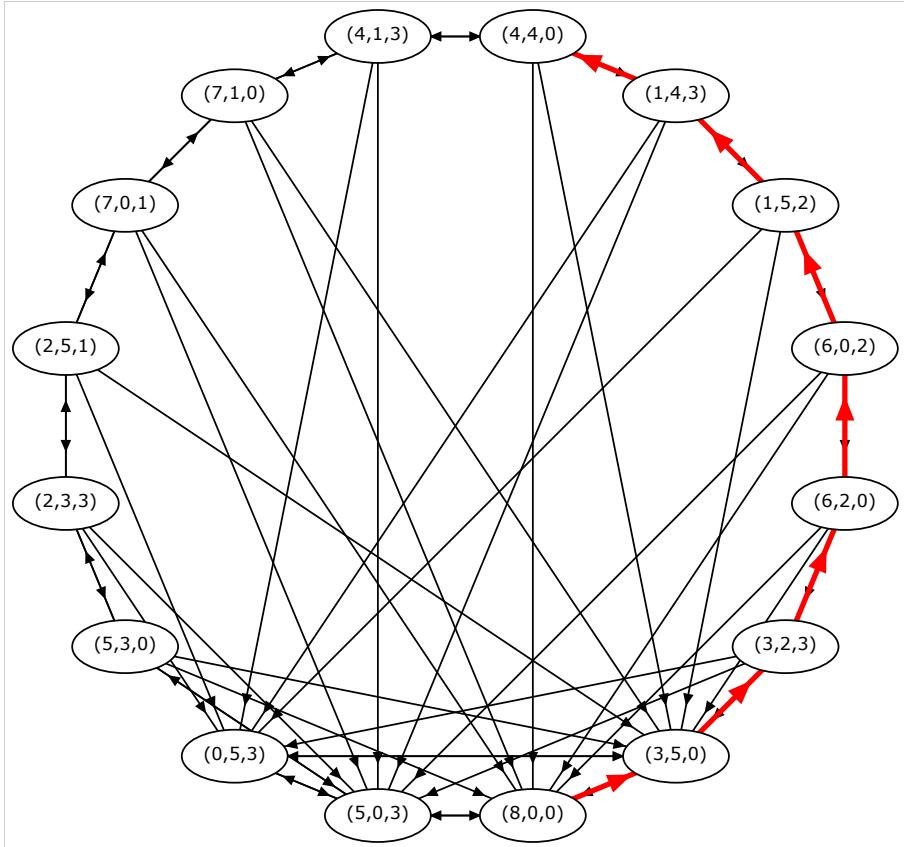


Figure 2.4: Solution of the 3-Jug Problem

2.3.5 Problem 5: The Fake Coin Puzzle I

Two weighings are sufficient! That is astonishing. First, the coins are numbered from 1 to 8. Of course, the first idea would be to weight all coins: 4 on one side of the balance and 4 on the other side. The fake coin is then at the lighter side, this is repeated with the four on the lighter side, etc. This method would lead to 3 weighings. But what if we only weight 6 coins initially? Coins 1,2,3 on one side of the balance and 4,5,6 on the other side? Because then we know: if the left side is lighter, the fake coin is there, if the right side is lighter, then it is on the right side, and if they are equal, then the fake coin must be 7 or 8. Bingo! That is the key idea: to separate the coins into three groups. This key idea is repeated in the second weighing. Suppose the left side with the coins 1, 2, 3 is lighter, then we know the fake coin is there. Now we weight only coin 1 against coin 2. If the left is lighter then the fake coin is 1, of the right is lighter then the fake coin is 2, otherwise the fake coin is 3, which was

not weighted! The whole weighings process is best shown in a Figure 2.5.

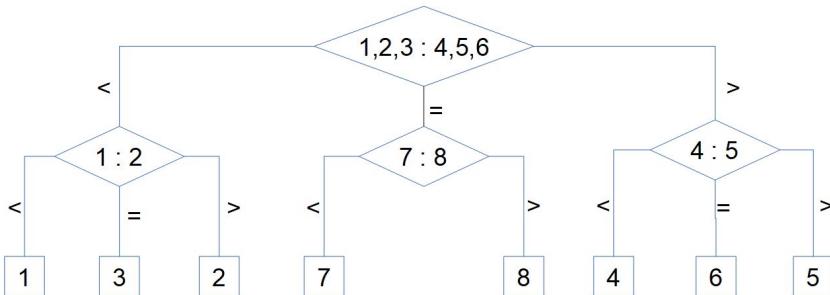


Figure 2.5: 8-coins Puzzle Solved

2.3.6 Problem 6: The Fake Coin Puzzle II

3 weighings are sufficient! This problem is an instructive example for *similarity*. Is this problem not similar to the previous one? Of course, it is! Can we use similar ideas? Let's try to apply the ideas from the the previous problem! Again, the coins are numbered from 1 to 12. First, we introduce a notation to represent the actual knowledge about a coin: the symbols $+$, $-$, and \pm following the coin number mean that the coin is “lighter”, “heavier” or “don’t know”, for example “ $1+$ ” means we know that the first coin is heavier if it is the fake coin. Similar to the previous problem, the first 4 coins are put on the left side of the balance and the next 4 coins on the right side, and 4 other coins are not touched. After the first weighing we know: if the left side is lighter, then the fake coin is within the first 4 coins and it is lighter *or* the fake coin is within the next 4 coins (coin 5 to 8) and it is heavier; if the two sides are equal, we can forget about all 8 coins and the fake coin is within the last 4 coins (not on the balance). We repeat the same idea, the weighings are shown in Figure 2.6 shows the solution.

Modeling by similarity is very common. One of the first question with a new problem is: “Do I know a similar problem?” Often the problem is different in many respects, but some key ideas can be reused.

2.3.7 Problem 7: The Horse Race Puzzle

This problem is sometimes asked in a technical interview question at companies like Google (better be prepared!) ([Youtube Video](#)). The problem can be solved by a systematical ordering of the facts and a clear grouping and regrouping of objects, no more is needed – these are other important tasks in

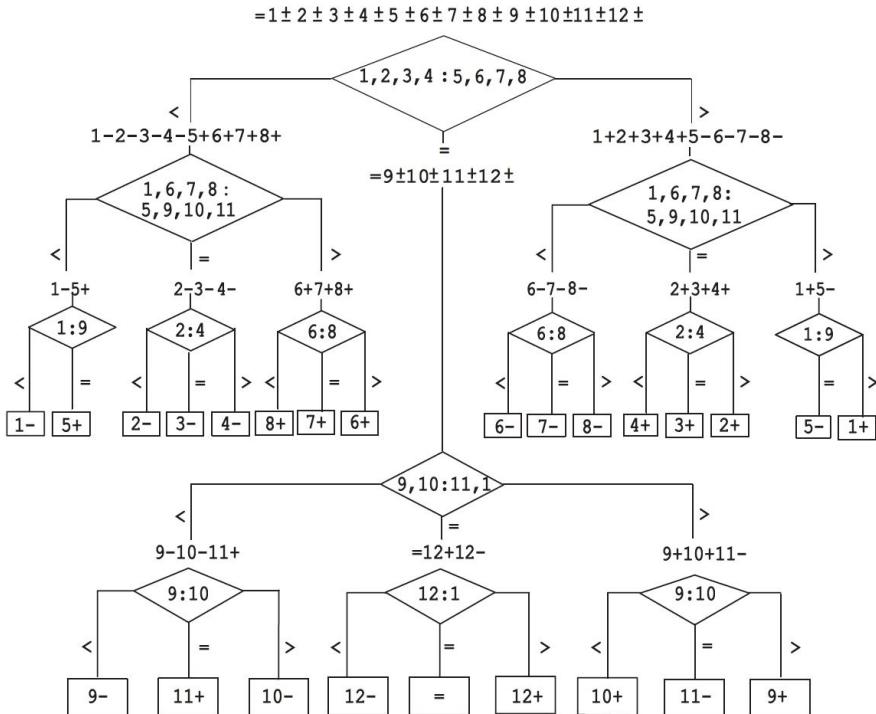


Figure 2.6: 12-coins Puzzle Solved [43]

modeling in general. *Order, sort, collect, and name* the components before doing more creative things in modeling. Often, the elements “fall into the right place” just by naming them systematically. In mathematical modeling the key element is often to identify the right *variables*.

Let’s analyze the horse problem: Only 5 horses can race at the same time, so:

1. Group the 25 horses into 5 groups of 5 horses. The groups are named a, b, c, d, e. Mark each horse with its group name. Hence, 5 horses are marked with an “a”, 5 with a “b”, and so on.
2. Let each group race, that is, the five horses in group a do the first race, then the five horses in group b, etc. **This gives 5 races.**
3. Mark each horse with its rang. Hence, the fastest horse in each group is marked with the number 1, the second fastest with number 2, etc. Each horse is now marked with its group name and its rank after the first five races.
4. Let the 5 horses with number 1 do another race (**race number 6**).
5. Order the 5 groups according to the fastest horses in that last race. So,

for instance, suppose $c1$ is the fastest horse, second is $b1$, third is $a1$, fourth is $d1$, and the slowest is $e1$ in this 6th race, then order the groups from top to bottom accordingly as shown in Figure 2.7 (first row contains all horses named c ; second row all horses named b , etc.).

$c5$	$c4$	$c3$	$c2$	$c1$
$b5$	$b4$	$b3$	$b2$	$b1$
$a5$	$a4$	$a3$	$a2$	$a1$
$d5$	$d4$	$d3$	$d2$	$d1$
$e5$	$e4$	$e3$	$e2$	$e1$

Figure 2.7: Ordering of the Groups

6. Clearly, the horse $c1$ is the fastest overall. Why? It is faster than any a horse (from the first race), and it is faster than any other 1 horse, and any 1 horse was faster than any 2 horse etc.
7. Who is second and third? In the group of the 1 horses only $b1$ and $c1$ can be second or third (but not $d1$ or $e1$). In group a only $a1$ could be third (all others in group a were slower). In group c and b the two fastest (beside $c1$) can be second or third. So let race the horses $c2, c3, b1, b2$, and $a1$ a final time (**race number 7**). The two fastest horses in that last race are second and third.

$c5$	$c4$	$c3$	$c2$	$c1$
$b5$	$b4$	$b3$	$b2$	$b1$
$a5$	$a4$	$a3$	$a2$	$a1$
$d5$	$d4$	$d3$	$d2$	$d1$
$e5$	$e4$	$e3$	$e2$	$e1$

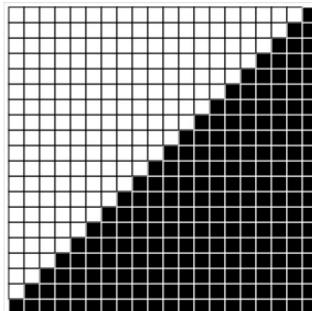
Figure 2.8: Ordering of the Groups

Another important concept in modeling appears here: We simplify by leaving out the unessential, but we simplify also by *implicitly assuming certain conditions*. In our example of the horse racing, we suppose that the horses perform

in the same way *in each race*, that is, the fastest in the first race is also the fastest in the 6th race, for instance. In all real problems, this is an *idealized* assumption. We know that this is not true. So, is the model then useless? Well, it depends on the context. In our horse-race example we may *define* the horse to be the fastest if it is the fastest in the 6th race, but it might have been slower than any other 1 horse in the 5 first races (when measured by a watch). In any case, it is important to be aware of all hidden assumptions of a model and in what context it would be inappropriate to apply the model.

2.3.8 Problem 8: Adding Numbers

$1 + 2 = 3, 3 + 3 = 6, 6 + 4 = 10,$
 $10 + 5 = 15, \dots$ I guess this is a too lengthy calculation. What if we pile small black squares, first one (on the top), then two, then three from top to bottom, etc. like in the Figure. Then we complete the piles with white squares to a larger square (of 20×20 in the Figure).



Now the problem can be reduced to the question: How many black squares are there? We have 20×20 total squares in this example, half of them plus half of 20 are black. Why? Look carefully, there are more black than white cells, because the diagonal also contains only black cells. If we made half of the diagonal (that is 10) white, then we would have exactly the same number of white and black cells. Hence, the number of black cells is: $20 \times 20/2 + 20/2 = 210$.

The same reasoning holds if there is a 1000×1000 square: Half of 1000^2 plus $1000/2$ are black, or in a 3×3 square, where $3^2/2 + 3/2$ are black – but wait a minute: $3/2$ is not an integer number. Well, $3^2/2$ is not integer either, but together they are integer (draw it, to verify). In general, for an $n \times n$ large square, the number of black squares – that is the sum from 1 to n – therefore is:

$$\frac{n^2}{2} + \frac{n}{2} = \frac{(n+1)n}{2}$$

This approach shows two important concepts in modeling: (1) Transform the problem into a very different (graphical) problem (piles of squares) that is structurally identical, (2) try first to pile 3, then 4, etc. Can you see the pattern? Can it be generalized?

A different approach is with arranging numbers. Try this in Figure 2.9. How many 1001 do we have? The answer is: 1000. Since each number is counted twice, we get: $1001 \cdot 1000/2$, which confirms the formula above.

1	+ 2	+ 3	+ 4	...	+ 999	+ 1000
+	1000	+ 999	+ 998	+ 997	...	+ 2
1001	+ 1001	+ 1001	+ 1001	...	+ 1001	+ 1001

Figure 2.9: Two Lines of Numbers

A digression: Proof that the formula is true for all $n > 0$. Proof by induction:

(1) prove that it is true for $n = 1$. This is the case, since:

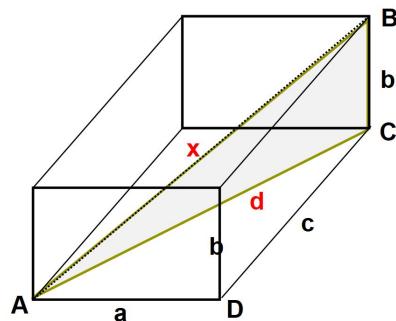
$$\sum_{i=1}^1 i = \frac{(n+1)n}{2} = \frac{2 \cdot 1}{2} = 1$$

(2) prove that if it is true for n then it is true for $n + 1$:

$$\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + n + 1 = \frac{(n+1)n}{2} + n + 1 = \frac{n^2 + 3n + 2}{2} = \frac{(n+2)(n+1)}{2}$$

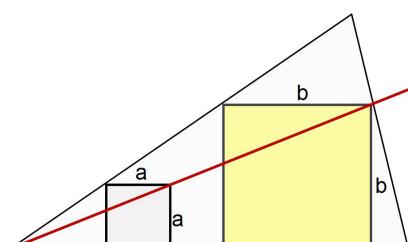
2.3.9 Problem 9: 3d-Diagonal

This problem shows another important concept in modeling: Reduce the problem to another known problem (Pythagoras). Add a line from A to C, then the triangle ABC form an auxiliary right-angled triangle where $x^2 = d^2 + b^2$. ADC forms another triangle for which holds: $d^2 = a^2 + c^2$. Eliminating d gives: $x = \sqrt{a^2 + b^2 + c^2}$.



2.3.10 Problem 10: Square Inside a Triangle

If you cannot see how to begin, just draw a square inside the triangle that shares a side with the triangle and touches one other side of the triangle. By drawing the red line, the solution pops up immediately. The key concept here is proportionality.



2.3.11 Problem 11: Birthday Paradox

For this problem, it is easier to calculate the probability that 2 persons do *not* have the same birthday.

- With 2 persons, the probability that they do not have the same birthday is $\frac{364}{365}$.
- With 3 persons the probability, that 2 persons do not have the same birthday, is $\frac{364}{365} \cdot \frac{363}{365}$.
- With 4 persons the probability, that 2 persons do not have the same birthday, is $\frac{364}{365} \cdot \frac{363}{365} \cdot \frac{362}{365}$. Do you see the pattern now?

Within a group of n persons the probability, that 2 persons do not have the same birthday, is therefore:

$$P'(n) = \frac{364}{365} \cdot \frac{363}{365} \cdot \dots \cdot \frac{365 - n + 1}{365}$$

This formula can be transformed to:

$$P'(n) = \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdot \dots \cdot \left(1 - \frac{n-1}{365}\right)$$

The initial problem that 2 persons out of n persons *do have* the same birthday is then $P(n) = 1 - P'(n)$. Figure 2.10 is a graph of this function $P(n)$. The graph shown that the probability that 2 persons in a group of 30 persons have the same birthday is more than 70%. In a party with 23 persons the probability is already larger than 50%.

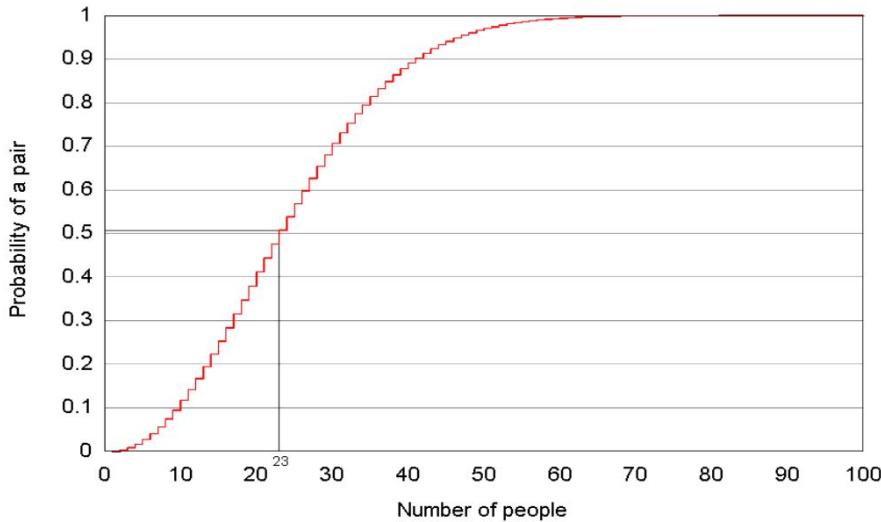
The birthday paradox problem has a very useful application in computer science for calculating the probability that 2 entries clashes in a hash table: Given a hash table of size 365, what is the probability that 2 entries clashes on the same memory location after n entries. This is a good example for the fact that the same model can have very different *interpretations* (or applications) in completely different domains.

Another aspect makes this problem interesting from a modeling point of view. The calculation of the expression for $P'(n)$ above is laborious. Isn't there a simpler model? Indeed we know that:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \quad (\text{and for } x \ll 1, e^x \approx 1 + x)$$

Let $x = -1/365$, then $x = -2/365, \dots$ then $P'(n)$ can be expressed as:

$$P'(n) = e^{-1/365} \cdot e^{-2/365} \cdot \dots \cdot e^{(1-n)/365}$$

Figure 2.10: Probability $P(n)$

$$= e^{-(1+2+\dots+(n-1))/365} = e^{(n^2-n)/730}$$

Simplifying and replacing by another approximation gives:

$$P(n) \approx 1 - e^{n^2/730}$$

Hence, the birthday problem can be approximated by a much simple model that is a very good approximation. One can easily calculate the probabilities using a hand calculator.

2.3.12 Problem 12: Length of Loch Ness Monster

30 meters is *not* correct! Let it do carefully. Suppose the length of the Loch Ness Monster is x meters, then this is the same as 20 meters plus half of x . So:

$$x = 20 + x/2 , \quad \text{and } x = 40$$

The Loch Ness Monster is 40 meters long.

2.3.13 Problem 13: Price of the Ball

Things are not as they seems to be at first glance, sit down and calculate! Let x be the price of a golf club, and let y be the price of the ball. Together they

cost 110, hence $x + y = 110$. The difference is 10, that means: $x - y = 10$. Now its is easy to see that $x = 105$ and $y = 5$. The price of the ball is 5 €(and not 10 €).

2.3.14 Problem 14: Postman Problem

What is the variable? “The time that the assistant alone takes in minutes”, name this quantity: x . It seems clear: The *difference* between “both doing the job” and “the postman alone does the job” is 25, hence, $x = 25$. Verify! Suppose we would ask for the postman’s time y alone given the assistant time 25 and together they take again 20. Using a symmetrical argument, we could then say that: $25 - y = 20$, or $y = 5$ which contradicts the fact that the postman alone takes 45 mins and not 5 mins! Therefore, *this approach is completely wrong!*

Well then, “Together they take 20 minutes, then *on average* each alone takes 40 minutes.” Since the postman really takes 45 minutes – he is slower then the average – the assistant must be faster then average, namely by $45 - 40 = 5$ minutes faster, hence: $x = 35$. This is again not correct! Why? Suppose together they would take only 10 minutes, then the average is 20 and the difference between the postman (45 minutes) and the average is 25. That means that the assistant would take $20 - 25 = -5$ minutes. That cannot be! Whatever short time they take together, the time of the assistant must *always* be positive. Therefore, *the approach is wrong again!*

We have to look at the problem from a *different angle*. So, let’s concentrate on the *number of letters distributed*, and not on the number of minutes. The right question here is: How many letters are distributed in a given amount of time by (1) the postman, (2) the assistant, (3) together. Surely, together they distribute the sum of the letters of each individually (supposing each works the same speed together or not together), that is, number of letters the postman distributes plus the number of letters the assistant distributes must be the numbers of the letters both distribute together in a given amount of time. Hence, we look at the number of letters distributed in one minute:

- In one minute, the postman distributes $1/45$ of all letters.
- In one minute, the assistant distributes $1/x$ of all letters.
- In one minute, together they distribute $1/20$ of all letters.

Now the model is easy to derive and to find our x , it is:

$$\frac{1}{45} + \frac{1}{x} = \frac{1}{20}$$

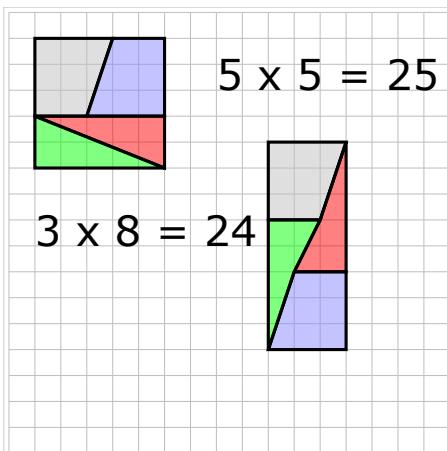
Resolving the equation gives $x = 36$. The assistant alone takes 36 minutes. Suppose, as an extreme case, together they would take only 1 minute, then the

assistant alone would take $45/44$ minutes (calculate!). This number is slightly higher than 1, which makes sense, because they together *always* do it in less time.

Well! One could argue that together they will chat a lot and wasting time or stop for a bite. This may happen, but then we have *another problem*. We made the silent assumption that their performance do not change when working together or working alone.

2.3.15 Problem 15: One Unit Missing

We take a smaller instance for the same problem (see Figure at the right). Now one can see immediately where the problem is: the diagonal in the 3×8 rectangle is *not* a straight line. Hence, none of the forms is a triangle. A quick calculation also shows that they *cannot* be triangles. The proportion of the height and the length of these triangles is $\frac{3}{8}$. At the position 5 at the length, the height of this “triangle” is 2. But we know that $\frac{3}{8} \neq \frac{2}{5}$. Hence, these forms are *not* triangles.



How did we discover these figures? They are “special numbers” 3×8 is almost 5×5 , and 8×21 is almost 13×13 . Sounds familiar? 1, 1, 2, 3, 5, 8, 13, 21, 34,... These are consecutive Fibonacci numbers. These numbers are defined by:

$$F_1 = 1, \quad F_2 = 1, \quad F_n = F_{n-1} + F_{n-2}, \quad \text{with } n \geq 2$$

The Fibonacci numbers have many interesting properties, one of them is

$$F_{n+1} \cdot F_{n-1} - F_n^2 = (-1)^n, \quad \text{with } n \geq 1$$

Proof (by induction):

1. The property is true for $n = 1$, since $1 \cdot 0 - 1^2 = (-1)^1$.

2. Supposing the property is valid for n , then it is valid for $n + 1$, namely we substitute F_{n-1} with $F_{n+1} - F_n$ in the property and we get:

$$F_{n+1}^2 - F_{n+1}F_n - F_n^2 = (-1)^n$$

By multiplying with -1 and transforming it we finally get:

$$F_{n+2} \cdot F_n - F_{n+1}^2 = (-1)^{n+1}$$

That proves the property.

3. In particular, for $n = 6$ we have: $13 \cdot 5 - 8^2 = (-1)^6$ producing the initial graph of the problem. We also have: $8 \cdot 3 - 5^2 = (-1)^5$.

2.3.16 Problem 16: How old am I?

Let us translate this problem into a mathematical formulation step by step, to remember the modeling process in such cases (see [20]):

1. *Understanding the problem:* What are we looking for? What do we know? What are the conditions? Are the data sufficient, are some data contradictory, irrelevant or redundant? Draw a figure! Introduce a suitable notation. Write it down. If you are stuck: begin again.
2. *Design a plan:* Do you know a similar problem? Find the connection between the data and the unknowns! Maybe you need to design auxiliary problems and intermediary steps! Look at the unknowns! Go back to the definitions! Decompose the problems and solve the parts! Did you use all data? Did you use all conditions?
3. *Carry out a plan:* Write it down step by step! Can you show that each step is correct? Make plausibility tests in each step. If possible simplify and do the calculations.
4. *Looking back:* Examine the solution. Can you check the result or the argument? Can you derive the result differently? Does the result make sense? Why? Why not?

Let's look at our problem now :

1. What are we looking for in our problem? "my age today"! This is unknown, so let us introduce the symbol a for "my age today". The symbol a stands for a positive number.
2. In the same way, let us introduce the symbol b for "my father's age today", the symbol c for "my age in ten year", and the symbol d for "my father's age in ten years".
3. Draw a figure (see Figure 2.11)

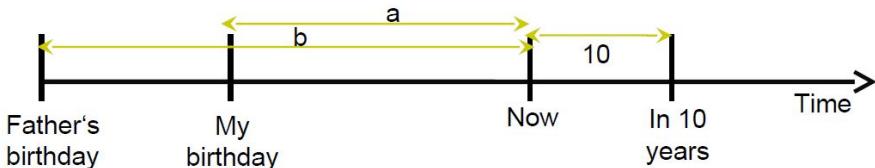


Figure 2.11: A Figure for the Problem

4. Connect the data with the unknown. It is clear from the statement that

$$c = a + 10 \quad \text{and} \quad d = b + 10$$

5. Furthermore: "My age is a third of my father's age" means that

$$a = b/3$$

6. Finally: "In 10 years my father will only be two times as old as I will be" gives:

$$d = 2c$$

7. Did we use all data and all conditions? I guess, yes!

8. We wrote down the complete problem in mathematics. Simplify it: eliminate the variables c and d gives the following:

$$a = b/3 \quad \text{and} \quad b + 10 = 2(a + 10)$$

9. Calculating gives: $a = 10$ and $b = 30$.

10. Looking back: My age is 10, in ten years I am 20, my father's age is 30, and in ten years he is 40. My father is three times older. Correct! But in ten years he is 40 and I am 20, so he is only two times older. Also correct! Everything is matching. So we are done.

2.3.17 Problem 17: 7 Digits Puzzle (redo)

The 7-digits puzzle is a good brain game. Interestingly, it could be formulated as a mathematical model. So how would one model this problem? First question: what are the variables? Answer: Which number to place in which circle. First of all, all the numbers must be different from each other and in the range $[1, 7]$. Hence, seven integer variables are introduced: x_i with $i \in \{1, \dots, 7\}$ for the 7 circles. For instance, x_1 is the number in circle A, x_2 is the number in circle B, and so on. Clearly, the three numbers in a line must be equal. There are five lines, so the constraints are as follows:

$$\begin{aligned}
 x_2 + x_4 + x_6 &= x_1 + x_4 + x_7 \\
 x_3 + x_4 + x_5 &= x_1 + x_4 + x_7 \\
 x_2 + x_5 + x_7 &= x_1 + x_4 + x_7 \\
 x_1 + x_3 + x_6 &= x_1 + x_4 + x_7 \\
 x_i &\in \{1, \dots, 7\}, \text{ all distinct}
 \end{aligned}$$

The model was implemented in the modeling language LPL and can be executed here: [Puzzle7](#).

2.3.18 Problem 18: The Clock Puzzle (redo)

This problem can also be formulated as a mathematical model. We want to known for each number on the clock's face which is on one side and which is on the other side. For this kind of situation, 0 – 1 variables are used – that is, variables that have only the value “zero” or “one” (or true and false). Let's introduce 12 binary variable x_i with $i \in \{1, \dots, 12\}$, with the following meaning: $x_i = 1$ if the number i is on one side (say side A), and $x_i = 0$ if the number i is on the other side (say side B). We have two conditions: (1) The sum of the numbers on side A must be half of all numbers. (2) Exactly two pairs of adjacent numbers must be on different sides. Suppose, one pair of numbers on different sides is 9 and 10. Then this means that x_9 and x_{10} must have different values, if one is 0 the other must be 1 and vice versa, (hence $x_9 \neq x_{10}$ would be a sufficient condition). This must be the case for *exactly two pairs* of numbers (The operator “*exactly(2)_i (E_i)*” the model below means: “exactly 2 out of all i expressions E_i are true”). Hence, the mathematical model for this problem is:

$$\begin{aligned}
 \sum_i i \cdot x_i &= \left(\sum_i i \right) / 2 \\
 \text{exactly}(2)_i (x_i \neq x_{i \bmod 12+1}) \\
 x_i &\in \{0, 1\}, \quad \text{forall } i \in \{1, \dots, 12\}
 \end{aligned}$$

The model was implemented in the modeling language LPL and can be executed here: [DrawClock](#).

2.4. Conclusion

This paper gives a short introduction to modeling with examples of how to model. The examples show that modeling is not trivial. Although the problems presented here are quiet simple, so to say, find a good model formulation is not always easy. How much more difficult must it be to formulate real-life problems with a lot of data and constraints. Modeling an art, sure, but a art

that can be learnt: one must practise. Continue modeling, look up my books with a lot of examples see [My Modeling Books](#).

CHAPTER 3

THE MODELING LANGUAGE LPL

“Practice makes perfect.”

This paper gives a first impression of the LPL modeling language. The paper is by no means complete or systematic. For learning LPL, I suggest to read through the tutorial [69] and to consult the reference manual [72]. Surely, after digesting this paper, the reader would be able to read and understand with ease most LPL models and would be able to write some basic models. A functionality overview of LPL can be downloaded at [65].

After an introduction to the basics, the paper exposes highlights several unique features of LPL such as modularity, the drawing library, logical constraint, and goal programming.

It would be an advantage, if the reader were familiar with the basics of some programming language to follow smoothly all the small models in this text.

3.1. Introduction

There are virtually hundreds of software and computer tools to implement various kinds of mathematical models. There exist special software for particular domains, or there are general tools for symbolic and algebraic manipulations or for numerical solutions. There are also extensions of modern established programming languages to formulate and solve models. Finally, various algebraic languages exist. Several options and tools are presented in a separate paper (see [74]).

LPL is an advanced modeling language and seems to me an interesting tool to start learning mathematical modeling. Its syntax is close to the common mathematical notational and at least its basics are easy and quick to learn. With LPL one can formulated small and large linear and non-linear models. It is linked to various free and commercial solvers. And one of its unique feature is to formulate discrete models using Boolean and logical operators. It is also ideal in an educational environment. Aside from that, large linear and integer models are also implemented and used by large companies such as ABB (see [extern/ABB-cpmPlus.pdf](#)) and SwissPort (see [extern/IFORS-News-2019-12-01.pdf](#)).

Furthermore, for testing and run your first models, there is no need to install any software on your computer, just use your favorite browser to start modeling. Later on, when you need serious error handling and interactivity in model building, an academic version of the LPL software can be downloaded for free and is installed in no time. It is shown later on in this paper.

Learning the basics of LPL is no wasted time. The knowledge is also useful to switch to other, and for your application more appropriate software or tools. So let's start!

3.2. The Basics

Here is a small model to be implemented in LPL:

$$\begin{aligned} \text{max } & 300x + 200y \\ \text{subject to } & 5x + 5y \leq 350 \\ & 6x + 2y \leq 300 \end{aligned}$$

The implementation in LPL of this model is straightforward:

```
model firstModel;
variable x; y;
maximize Obj: 300*x + 200*y;
constraint C: 5*x + 5*y <= 350;
              D: 6*x + 2*y <= 300;
Writep(x,y);
end
```

1. Every model begin with the keyword **model** followed by an identifier and a semicolon and ends with keyword **end**.
2. The LPL model consists of five declarations and a statement.
3. Two variable declaration with the name x and y begin with keyword **variable**. If the same declaration repeats, the keyword can be dropped.
4. Two constraint declarations with name C and D followed by a colon and an expression. Note that constraints always have a name.
5. A maximizing declaration with name Obj also followed by a colon and an expression.
6. Finally, a statement, a function call *Writep* to print the two variables with their value.

Copy this model and paste it here: [empty](#) (A browser opens). Then click on the button *Run and Solve*. A moment later, the solution of this model is displayed: $x = 40, y = 30$.

To formulate larger models with thousands of variables and constraints, we use a notation in mathematics that is called *indexed notation*¹. The following is a linear model with $m > 0$ constraints and $n > 0$ variables.

¹ If the reader is not familiar with indexed notation, study the following paper [67].

$$\begin{aligned}
 & \text{max} && \sum_{j \in J} c_j x_j \\
 & \text{subject to} && \sum_{j \in J} a_{i,j} x_j \leq b_i \quad \text{forall } i \in I \\
 & && x_j \geq 0 \quad \text{forall } j \in J \\
 & \text{with} && I = \{1, \dots, m\}, \quad J = \{1, \dots, n\} \quad m, n \geq 0
 \end{aligned}$$

One of the main strength of LPL is to use the index notation. Data can be declared by parameters :

```

parameter m := 1000;
            n := 2000;

```

The keyword **parameter** starts a parameter declaration. A name follows and optional a assignment. It means that **m** gets a value of 1000, and **n** gets a value of 2000.

Index sets also are a fundamental concept in larger models. LPL declares index sets using the keyword **set** declaring the sets **I** and **J** in the same way as above in the mathematical notation :

```

set I := 1..m;
      J := 1..n;

```

In the parameter declaration, not only singleton data can be defined also vector data, or matrices, or higher dimensional data. The vectors b_i with $i \in I$, c_j with $j \in J$, and the matrix $a_{i,j}$ are declared (without assignment) :

```

parameter b{i in I};
            c{j in J};
            a{i in I, j in J};

```

These parameters can be assigned in the same way. The assignment can take place directly at the declaration or later on in a proper assignment as in :

```

a{i in I, j in J} := if(Rnd(0,1)<0.02 , Rnd(0,60));
c{j in J}           := if(Rnd(0,1)<0.87 , Rnd(0,9));
b{i in I}           := if(Rnd(0,1)<0.87 , Rnd(10,70000));

```

The assignment is done through an expression that generates random numbers. The function **if**(cond, exp) returns a value defined by exp if the Boolean condition cond is true else it returns zero. (It is the same as the ternary operator $c ? a : b$ in some programming languages as C, Java, Python, and others.) The function **Rnd(a, b)** returns a random number uniformly distributed between a and b. Hence, the first assignment generates first a random number between 0 and 1, and if it is smaller than 0.02 then a second random number between 0 and 60 is generated and assigned to the

matrix else 0 is assigned. This operation is repeated for all elements in I combined with all elements in J. The statement is similar to a double loop in a programming language like C (suppose `myrand()` returns a double between 0 and 1) :

```
double a[][]; int i,j;
for (i=0; i<m; i++) {
    for (j=0; j<n; j++) {
        a[i,j] = myrand() < 0.02 ? 60*myrand() : 0;
    }
}
```

In the same way as the loops above, the LPL statement

```
| parameter a{i in I, j in J} := Rnd(0,1);
```

runs through the sets in lexicographical order, that is, right-most index first, left-most index last, and on each pass it assigns a random value to the matrix entry.

In other words, the matrix a contains about 2% of data that are different from zero. The large majority of its elements is zero. Of course, LPL only stored the non-zeroes (in a sparse way).

A last remark about a simplified notation in LPL is notable: Although the previous notation as in `a{i in I, j in J}` is perfectly legal syntax in LPL, a shorter notation is preferable. In LPL set names are normally in lowercase letters and can be used as indexes too. There is generally no need to use a separate symbol for sets. So the previous declarations could also simply be written as:

```
set i := 1..m;
      j := 1..n;
parameter a{i,j};
            c{j};
            b{i};
```

In the same way as parameters, also variables (`x{i}`) and constraints `C{i}` can be indexed to specify a vector or a higher dimensional quantity of single objects. The same indexing mechanism can also be applied to several index operators, such as $\sum_i \dots$ (in LPL `sum{i} ...`).

Now all elements are ready to formulate the general linear model with 1000 constraints and 2000 variables :

```
model largerModel;
parameter m := 1000; n := 2000;
set i := 1..m; j := 1..n;
parameter
    a{i,j} := if(Rnd(0,1)<0.02 , Rnd(0,60));
    c{j}   := if(Rnd(0,1)<0.87 , Rnd(0,9));
```

```

    b{i} := if(Rnd(0,1)<0.87 , Rnd(10,70000));
variable x{j};
constraint C{i}: sum{j} a[i,j]*x[j] <= b[i];
maximize Obj: sum{j} c[j]*x[j];
Write('Objective Value = %7.2f \n', Obj);
Write{j|x|}' x%-4s = %6.2f\n' , j,x);
end

```

Again, copy this model and paste it here: [empty](#) (A browser opens). Then click on the button *Run and Solve*. A moment later, the solution of this model is displayed:

```

Objective Value = 77599.07
x22   = 34.64
x31   = 14.46
x69   = 110.61
... about 85 more values ...

```

About 90 variables out of the 2000 have a value different from zero. Three are shown in the list above.

Note that the function **Write** was used to write a formatted output of the solution. This function is a very powerful method to generate text output, sophisticated reports using the library of [FastReport](#), as well as database tables or even Excel sheets. See the reference manual for more information [72].

Data can also be directly added to the model. As an exercise, the simple example above is written in an indexed notation:

```

model secondModel;
set i := [1 2];
      j := [a b];
parameter b{i} := [350 300];
      c{j} := [300 200];
      a{i,j} := [5 5 , 6 2];
variable x{j};
constraint C{i}: sum{j} a[i,j]*x[j] <= b[i];
maximize Obj: sum{j} c[j]*x[j];
Writep(x);
end

```

(As before, copy this model, paste it here: [empty](#) and run it.) Inline data tables are enclosed in [...] and are evaluated and assigned before any other assignment. The elements are listed in lexicographical order. Note that set elements in this case are strings.

Another feature in LPL is repeated execution and branching in its execution. The two statements for looping begin with **for** and **while** and for branching it begins with **if**. The syntax is:

```
for{setName} do
    ...statement list...
end;

while expr do
    ...statement list...
end

if expr then
    ...statement list...
else           //optional part
    ...statement list...
end
```

The following program implements the greatest common divisor of two numbers (the algorithm is quite inefficient – there are much better methods², but it shows a loop and a branching statement):

```
model gcd;
integer a := 1943; b := 2813;
c := if(a<b, a, b);
d := 1;
for{i in 1..c} do
    if a%i = 0 and b%i = 0 then d:=i; end
end
Write('The gcd of %d and %d is %d\n', a, b, d);
end
```

(Copy this model, paste it here: **empty** and run it.) Note first, that the assignment operator is `:=` (not `=` like in C, Python, Java etc.) and the Boolean equal operator is `=` (not `==`). `%` is the modulo operator. The keyword **integer** start a parameter declaration that is integer. It is a shortcut for **integer parameter**. **if** has two different uses: the first was already explained above and is a function. The second starts a branching statement. Finally, `{i in 1..c}` declares an anonymous set with a local index `i`. (Another implementation of the Euclid algorithm is given in **euclid**.)

The example shows that LPL is a complete programming language and is Turing complete, in order to be able to pre- and postprocess data before and after solving a model. However, it is far from be a modern programming

² This method loops through ALL integers `i` from 1 to the smaller number (`a` or `b`) and checks whether `a` and `b` is divisible by `i`, and if it is it retains the last found number. A better way is the Euclid algorithm.

language – in its actual version³. It lacks object oriented concept; its memory is not stack-based, that is, although one can define user defined functions (see below), the function cannot be called recursively; it is interpreted and a just-in-time compiler is missing; so it is not very efficient for most algorithmic tasks. Nevertheless, LPL does something very well: large models, eventually with sparse data sets, can be efficient and compactly formulated and ran. It links well to several commercial and free solver libraries.

3.3. Sub-models

Models can be organized into sub-models each with its own name space, and they can be distributed into several file. Let's give a small simple example that shows also how the data can be separated from a model structure by using the two linear models above (open `subModels` to run it, modify it by assigning 2 to the parameter `selectData`):

```

model subModels;
  parameter selectData:=1;
  if selectData=1 then data1; else data2; end;
  set i; j;
  parameter a{i,j}; c{j}; b{i};
  variable x{j};
  constraint C{i}: sum{j} a[i,j]*x[j] <= b[i];
  maximize Obj: sum{j} c[j]*x[j];
  Write('Objective Value = %7.2f \n', Obj);
  Write{j|x|}'    x%-4s = %6.2f\n' , j,x);

model data1;
  i := [1 2];
  j := [a b];
  b{i} := [350 300];
  c{j} := [300 200];
  a{i,j} := [5 5 , 6 2];
end;

model data2;
  parameter m := 1000; n := 2000;;
  i := 1..m; j := 1..n;
  a{i,j} := if(Rnd(0,1)<0.02 , Rnd(0,60));
  c{j} := if(Rnd(0,1)<0.87 , Rnd(0,9));
  b{i} := if(Rnd(0,1)<0.87 , Rnd(10,70000));
end;
end
```

The code contains the main model `subModels` and two sub-models `data1`

³ My vision of a fully-fledged modeling language are exposed in another paper, see [70].

and `data2`. The main model does not contain data, only the declarations. One of the sub-model is called and ran from the main model, depending on the value of the parameter `selectData`⁴. Note the double semicolon in the sub-model `data2`. `m` and `n` are two local parameters and after their declaration follows a double semicolon. This is important in this context because it terminates the declaration with an empty statement (a semicolon, because the next `i := ...` is not local, it is an assignment to the global set `i`. The data model could also be in another file. To simplify the presentation, the data – in fact two data-sets – are included within the main model.

3.4. Drawings

Specially for the presentation of the results, figures and drawings are very helpful to understand the solution of a model. Therefore, LPL contains a small drawing library to generate an vector graph in SVG format. The following model show the use of this library (open `chess5b` to run the model and to see the graphic) :

```
model chess;
set i, j := [1..8];
integer variable x{i} [1..#i];
constraint
  D: Alldiff({i} x[i]);
  S: Alldiff({i} (x[i]+i));
  T: Alldiff({i} (x[i]-i));
solve;
parameter X{i,j} := if(j=x[i],1);
Draw.Scale(50,50);
{i,j} Draw.Rect(i,j,1,1,if((i+j)%2,0,1),0);
{i,j|X} Draw.Circle(j+.5,i+.5,.3,5);
end
```

This model implements a placement of 8 queens on a chessboard which cannot attack each other. The model shows some more interesting features:

1. The same set definition has *two* names: `i` and `j` (they are separated by a comma, not by semicolon). This is handy because they are used for the rows and the columns on the chessboard.
2. The variable are declared as integer variable in the range `[1..8]`. (`#i` is a notation for the cardinality of the set `i`). For example, $x_2 = 3$ means that the queen on the second row should be placed in column 3.
3. The constraints implements a global constraint called `Alldiff` meaning that all the variables must be different from each other. (It goes too far here, but an `Alldiff` constraint is either passed directly to a solver

⁴ Note also that identifiers can be used before they are declared.

like hexaly solver, or it is automatically translated by LPL to integer linear constraints – for a linear solver like Gurobi.)

4. The `solve` statement says to solve the constraints and return (any) possible solution.

After the solution has been found, a graphic is generated using the library `Draw`. This library contains a dozens of functions such as `Scale` (for the $x - y$ sizing), `Rect` (to draw a rectangle), `Circle` (to draw an circle), etc. (Note that `{i, j}` is a shortcut for `for{i, j} do ... end;`). Hence, 8×8 rectangles are drawn at position (i, j) with size $(1, 1)$ (multiplied by 50 pixels), alternating with color black (0) and white (1) forming the chessboard grid. Then a circle is drawn on each field with color blue (5) if $x_{i,j}$ is 1 (8 of them are 1).

3.5. Logical and Integer Modeling

One of the unique feature of LPL is its capability to use logical operators to model constraints. A small example is given as follows (see also `logic4` for further explanation) :

```
model Logic4;
  variable x [0..10]; y [0..20];
  constraint Grey: (x<=5 and y<=2) or (y<=x and y<=6-x);
  maximize obj: x+y;
  Write('Optimum is: (%d,%d)\n' ,x,y);
end
```

Two variables x and y with lower bounds 0 and upper bound 10 and 20 are declared. The constraint contains arithmetic and Boolean operators defining a concave set. Besides `and` and `or`, LPL also defines additions Boolean and logical indexed operators, such as `not`, `nand`, `implication`, and indexed `and`, `or` `atleast`. For more information, see the paper [68], and the reference manual [72].

3.6. Conclusion

This paper gives a first impression on the modeling language LPL. It is useful in a educational context to model all kinds of models: linear, non-linear, permutation problems. It is a full-fetched programming language. It can be used in a commercial context for formulating large linear integer problems that can be solved by commercial solvers like Gurobi, Cplex, or Xpress. For more learning the language see the tutorial and the reference manual as already mentioned.

CHAPTER 4

INDEX NOTATION IN MATHEMATICS

“Learn to walk before you run.”

This paper explains indexing notation in mathematics and its implementation in the modeling language LPL. Indexing is one of the most fundamental concept in mathematical notation. It is extremely powerful and allows the modeler to concisely formulate large and complex mathematical model. A certain number of exercises are given and implemented in LPL, so the reader can test the concepts.

In my experience, many students fail to create – or even read – mathematical models, because they didn't learn correctly how to use indexing notation. It is not particularly difficult and it can be learnt easily. Unfortunately, in most mathematical modeling textbook this is badly neglected. Hopefully, this paper can remedy deficit.

4.1. Introduction

To work with a mass of data, a concise formalism and notation is needed. In mathematics, the so-called *indexed notation* is the appropriate notation. This notation is an integral and fundamental part of every mathematical modeling activity – and they are also used in mathematical modeling languages as in LPL [76] – to group various objects and entities. It is surprising how little has been published in the community of modeling language design on this concept. Even in mathematical textbooks, the indexed notation used in formulae is often taken for granted and not much thought is given to it¹. This fact contrasts with my experience that students often have difficulties with the indexed notation in mathematics.

To represent single data, called *scalars*, names and expressions are used, such as:

$$x, \quad y \quad \text{or} \quad x - y$$

To specify further what we mean by a symbol, we write $x, y \in \mathbb{R}$, for example, saying that x and y represent any real numbers. We can attach a new symbol (z) to an expression as in:

$$z = x - y$$

meaning that z is a new number which is defined as the difference of x and y . We say “ z substitutes the expression $x - y$ ”. In this way, we can build complex expressions. We learnt this algebraic notation already in school. These symbols have no specific “meaning” besides the fact that they represent numbers.

¹A notable exception is [23], which devotes the whole Chapter 2 to indexed expressions and their use in mathematics.

To use them in a modeling context, we then attach a meaning. For example, in an economical context we might interpret “ x ” as “total revenue” and “ y ” as “total costs”. Then “ z ” might be interpreted as “total profit”. There is nothing magic about these symbols, instead of writing $z = x - y$ we can also write:

$$\text{total profit} = \text{total revenue} - \text{total costs}$$

However, to economize our writings we prefer short “names”, like x , y , and z . But there is nothing wrong with longer names to make expressions more readable in a specific context.

It seems to be a little bit less familiar that in the same way as using symbols for *scalars*, symbols can represent mass of data. Suppose that we want to express the profit of *several* profit centers in a company. Then we could write:

$$\begin{aligned}\text{profit at center 1} &= \text{revenue at center 1} - \text{costs at center 1} \\ \text{profit at center 2} &= \text{revenue at center 2} - \text{costs at center 2} \\ \text{profit at center 3} &= \text{revenue at center 3} - \text{costs at center 3} \\ &\dots \quad \text{etc.} \quad \dots\end{aligned}$$

or (using the “names” x_1 , x_2 , y_1 , and so on)

$$\begin{aligned}z_1 &= x_1 - y_1 \\ z_2 &= x_2 - y_2 \\ z_3 &= x_3 - y_3 \\ &\dots \quad \text{etc.} \quad \dots\end{aligned}$$

The “etc.” means that we have to continue writing as many lines as we have profit centers, there may be dozens – a rather boring task!

There is, however, a much more economical way in mathematics to represent *all* these expressions in one single statement. It is the *indexed notation*. To formulate them, we first introduce a set: the set I of all profit centers:

$$I = \{\text{center 1}, \text{ center 2}, \dots\}$$

here again the “...” means that we continue writing all centers in a list. Often a short name form is used as follows:

$$I = \{1 \dots n\} \quad \text{where } n \in \mathbb{N}^+$$

In the previous set definition we just mean that there are n centers, n being a positive number. We are not concerned right now of how many centers there

are, we just say n – it can be 5 or 1000. Of course, in a concrete context we must specify the number n , but it is part of the data of that specific context.

In a second step, we introduce symbols for all profits, revenues and costs. Now, instead of using each time a new symbol, we just attach an subscript to the symbols: z_i , x_i , y_i to express the fact that they “mean” the profit, the revenue and the cost of the i -th center, together with the notation $i \in I$, which means that i just designates an arbitrary (i -th) element in I . Hence, All data can be written in a concise way as follows:

$$x_i, \quad y_i, \quad z_i \quad \text{where } i \in I$$

Mathematically speaking, three *vectors* are declared to represent all data. The list of expression then can be written in a single statement as follows:

$$z_i = x_i - y_i \quad \text{with } i \in I$$

This notation is in fact nothing else than an economical way of the following n expressions:

$$\begin{aligned} z_1 &= x_1 - y_1 \\ z_2 &= x_2 - y_2 \\ &\dots \\ z_n &= x_n - y_n \end{aligned}$$

where n is some positive integer.

In a similar way, we can concisely build an expression that sums *all* profits, for instance. For this purpose we use the mathematical operator \sum , the Greek sigma symbol. The total profit p of all profit centers can be formulated as follows:

$$p = \sum_{i \in I} z_i$$

Again, this formmula is nothing else than a shortcut for the long expression:

$$p = z_1 + z_2 + \dots + z_n$$

After this introductory example, this paper presents now a more precise way, on how the indexed notation is defined and how it can be used in modeling.

4.2. Definitions and Notations

Index sets are essential for mastering the complexity of large models. All elements in a model, such as variables, parameters, and constraints – as will be shown later on – can appear in groups, in the same way they are indexed in mathematical notation.

The convention in algebraic notation to denote a set of similar expressions is to use *indexes* and *index sets*. For example, a summation of n (numerical) terms

$$a_1 + a_2 + \cdots + a_{n-1} + a_n \quad (4.1)$$

is commonly abbreviated using the notation

$$\sum_{i=1}^n a_i \quad (4.2)$$

The expression (1) is called *three-dots notation* and expression (2) is called *sigma-notation*. Both expression are equivalent, but (2) is much shorter and more general. The later was introduced by Joseph Fourier in 1820, according to [23, p. 22]. There exist different variants of expression (2) :

$$\sum_{1 \leq i \leq n} a_i , \quad \sum_{i=1}^n a_i , \quad \sum_{i \in \{1 \dots n\}} a_i , \quad \sum_{i \in I} a_i \quad \text{with } I = \{1 \dots n\} \quad (4.3)$$

All four expressions in (3) are equivalent and they have all their advantages and disadvantages. The last notation in (3) is more general, because the index set I can be an arbitrary set defined outside the summation expression.

4.2.1 Exercises in the Sigma Notation

Before we generalize the indexed notation, let us give some examples with the sigma form.

A notation $\sum_{i=1}^5 x_i$ means that i is replaced by whole numbers starting with 1 until the number 5 is reached. Thus

$$\sum_{i=2}^4 x_i = x_2 + x_3 + x_4$$

and

$$\sum_{i=2}^5 x_i = x_2 + x_3 + x_4 + x_5$$

Hence, the notation $\sum_{i=1}^n$ tells us:

1. to add the numbers x_i ,
2. to start with $i = 1$, that is, with x_1 ,
3. to stop with $i = n$, that is, with x_n (where n is some positive integer).

As an example, let us assign the following values: $x_1 = 10$, $x_2 = 8$, $x_3 = 2$, $x_4 = 15$, and $x_5 = 22$. Then we have:

$$\sum_{i=1}^5 x_i = x_1 + x_2 + x_3 + x_4 + x_5 = 10 + 8 + 2 + 15 + 22 = 57$$

The name i is a dummy variable, any other name could be used. We could have used j , the expression would have been exactly the same, hence:

$$\sum_{i=1}^5 x_i = \sum_{j=1}^5 x_j$$

Now let us find $\sum_{i=1}^4 3x_i$ based on the previous values. Again we start with $i = 1$ and we replace $3x_i$ with its value:

$$\sum_{i=1}^4 3x_i = 3x_1 + 3x_2 + 3x_3 + 3x_4 = 3 \cdot 10 + 3 \cdot 8 + 3 \cdot 2 + 3 \cdot 15 = 105$$

Similarly, let us find $\sum_{i=2}^5 (x_i - 8)$: This is:

$$\begin{aligned} \sum_{i=2}^5 (x_i - 8) &= (x_2 - 8) + (x_3 - 8) + (x_4 - 8) + (x_5 - 8) \\ &= (8 - 8) + (2 - 8) + (15 - 8) + (22 - 8) = 15 \end{aligned}$$

One should be careful with the parentheses. The expression $\sum_{i=2}^5(x_i - 8)$ is not the same as $\sum_{i=2}^5 x_i - 8$. The later evaluates to (the 8 is not included in the sum):

$$\sum_{i=2}^5 x_i - 8 = x_2 + x_3 + x_4 + x_5 - 8 = 39$$

We also use sigma notation in the following way:

$$\sum_{j=1}^4 j^2 = 1^2 + 2^2 + 3^2 + 4^2 = 30$$

The same principle applies here. j is replaced in the expression j^2 by numbers starting with 1 and ending with 4, and then adding up all four terms.

For the sigma notation we have three important transformation rules:

Rule 1: if c is a constant, then:

$$\sum_{i=1}^n cx_i = c \sum_{i=1}^n x_i$$

Rule 2: if c is a constant, then:

$$\sum_{i=1}^n c = nc$$

Rule 3: Adding the term can be distributed:

$$\sum_{i=1}^n (x_i + y_i) = \sum_{i=1}^n x_i + \sum_{i=1}^n y_i$$

Proof:

$$\begin{aligned} \sum_{i=1}^n cx_i &= cx_1 + cx_2 + \dots + cx_{n-1} + cx_n \\ &= c \cdot (x_1 + x_2 + \dots + x_{n-1} + x_n) = c \cdot \sum_{i=1}^n x_i \end{aligned}$$

$$\sum_{i=1}^n c = \underbrace{c + c + \dots + c}_{n\text{-times}} = n \times c = nc$$

$$\begin{aligned} \sum_{i=1}^n (x_i + y_i) &= (x_1 + y_1) + \dots + (x_n + y_n) \\ &= (x_1 + \dots + x_n) + (y_1 + \dots + y_n) = \sum_{i=1}^n x_i + \sum_{i=1}^n y_i \end{aligned}$$

End of proof

4.2.2 Formal Definition

More formally, an expression written in *indexed notation* can be noted as follows:

$$\bigodot_{i \in I} E_i \quad (4.4)$$

It consists of the following elements:

1. An *index operator*, here written as \bigodot . This operator can be any associative and commutative operator, such as \sum (summation), for example.
2. An *index set*, represented by I , being any countable (finite or infinite) collection of *elements*. The elements can be any indivisible item. Often they are integers.
3. An *active index* i (in $i \in I$), attached to the index operator.
4. An *indexed expression* E_i , that normally contains the same index name again. We call this index *passive index*. The reason for that will be clear later on.

(1) The index operator: It is clear why the *index operator* must be commutative and associative: A mathematical set is unordered and its elements can be “traversed” in any order. As an example, the \sum does not specify in which order the indexed expressions E_i are added up. Various operators fulfill these requirements. Besides the addition, represented by the operator \sum , the following index operators are commonly used (examples will be given later on):

- The product (multiplication) \prod : The indexed expressions are multiplied.
- The Max (Min) operator, written as $\max_{i \in I}$ ($\min_{i \in I}$): They return the largest (smallest) indexed expression in the list.
- In Boolean logical expressions we use the AND- and OR-operators (\wedge , \vee): They check whether all indexed expressions are true or at least one is true.
- A particular index operator is the *list operator* $\dot{\wedge}$. It is used to list indexed expressions (see later on).

(2) The index set: In the simplest case, the elements of the *index set* are integers, as in :

$$I = \{1, \dots, n\}, \quad \text{where } n > 0$$

However, they can also be identifiers (symbolic names), strings or any other items and even sets representing elements of a set. In the following index set I with the four elements are identifiers :

$$J = \{ \text{spring}, \text{summer}, \text{autumn}, \text{winter} \}$$

Ordering: Sets are unordered in mathematics as mentioned above. In modeling environments, however, we often must impose a specific order, for example, if a set consists of a number of time periods or time points. Then the natural order is the sequence of these periods. For example, the weekdays:

$$K = \{ \text{Mon}, \text{Tue}, \text{Wen}, \text{Thu}, \text{Fri}, \text{Sat}, \text{Son} \}$$

Example: Suppose that a_i is the quantity of sells in time period $i \in I$. To know the difference d_i from one time period to the next, we might use the expression:

$$d_i = a_{i+1} - a_i \quad \text{forall } k \in \{1, \dots, n-1\}$$

(Note that the last value d_n is not defined!).

In modeling, the order of index sets is important in such a context, because we often need to refer to the “previous”, the “next”, the “last” or the “first” period in an indexed expression. In spacial arrangements the order might also be important. Consider a chessboard with $I = \{1 \dots 8\}$ rows and columns, then it makes sense to refer to the “neighbor” columns at the left or right and the “neighbor” rows above or below a given cell. In another context, the ordering might have a semantical meaning. For example, if we want – for a specific purpose – impose an order on a list of products. We can order the product by “importance” or whatever criterion we need. Without ordering the statement to generate a “list with the first 10 products” would have no meaning. Without any externally imposed criterion the ordering of the products would have no meaning, as in:

$$I = \{ \text{bred}, \text{cheese}, \text{meat}, \text{eggs} \}$$

Tuples: The elements of an index-set may also be tuples, which is extremely common in modeling. A tuple is an ordered list of *components* which are separated by a comma and surrounded by parentheses. For example “(lion, mammal)” is a tuple that may express the fact that a lion belongs to the category of mammals. A list of such tuples can build a (large) index-set:

$$J = \{ (\text{lion}, \text{mammal}), (\text{sparrow}, \text{bird}), (\text{snake}, \text{reptile}), (\text{cow}, \text{mammal}), \dots \}$$

Another example is the (infinite) list of all positive integer grid points in a Euclidean space, written as :

$$\begin{aligned} I &= \{(x, y) \mid x, y \in \mathbb{N}^+\} \\ &= \{(0, 0), (1, 0), (1, 1), (0, 1), (0, 2), (1, 2), (2, 2), (2, 1), \dots\} \end{aligned}$$

Set of sets: Another important case in modeling is “set of sets”. As a concrete example, certain clients are served from certain warehouses. Let $I = \{1, \dots, m\}$ be a set of warehouses and let $J = \{1, \dots, n\}$ be a set of clients, then the set S :

$$S = \{Q_i \mid i \in I, Q_i \subseteq J\}$$

This set S tells us which client $j \in J$ is served from which warehouse $i \in I$. Note that the elements of the index-set S are themselves index-sets (Q_i)

Example:

$$S = \{\{1, 2, 3\}, \{2, 3\}, \{\}, \{2\}\} \quad , \text{ hence} \quad \begin{aligned} Q_1 &= \{1, 2, 3\} \\ Q_2 &= \{2, 3\} \\ Q_3 &= \{\} \\ Q_4 &= \{2\} \end{aligned}$$

says, that warehouse 1 serves client 1, 2, 3, warehouse 2 serves clients 2, 3, warehouse 3 do not serve any client, and warehouse 4 serves client 2, that is, Q_i is the set of clients served by warehouse $i \in I$.

Alternatively, this kind of data (set of set) can also be formulated as a tuple list:

$$S = \{(i, j) \mid (i, j) \in \{(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (4, 2)\}\} \quad \text{forall } i \in I, j \in J$$

(3) The active index: The *active index* is a (dummy) name (or an identifier) representing an arbitrary element in the index set. The *passive index* is also an identifier used in the indexed expression which must have the same spelling as the active index. The term $i \in I$, consisting of the active index and the index set, is called an *indexing term*. The active index and the passive index are used as place-holders to define a bijective mapping between the index set and a *set of indexed expressions*. Each element i in the index set maps to an indexed expression by replacing the passive index with the corresponding element i in the index set. This mapping is called the *index mechanism*. In fact, it defines the (bijective) function $f : i \xrightarrow{f} E_i$ with $i \in I$. This mapping supposes that

the corresponding indexed expressions E_i exist. The *domain* of this function f contains the elements of the index set $\{1, 2, \dots, n\}$, and the *codomain* (or the *range*) is the set of indexed expressions $\{E_1, E_2, \dots, E_n\}$.

(4) The index expression: The *indexed expression* is an arbitrary expression. It is not necessary that it contains a passive index. For example, in the expression:

$$\sum_{i \in I} 3 \quad \text{with } I = \{1, \dots, n\}$$

the indexed expression is just 3. The whole expression then reduces simply to:

$$\underbrace{3 + 3 + \dots + 3}_{n-\text{times}} = 3 \cdot n$$

The indexed expression can also be reduced to a passive index as in:

$$\sum_{i \in I} i \quad \text{with } I = \{1, \dots, n\}$$

in which case the indexed expression is just i . The whole expression reduces to:

$$1 + 2 + \dots + n$$

The indexed expression can itself contain index operators. For example:

$$\sum_{i \in I} \left(\sum_{j \in J} F_{i,j} \right) \quad \text{forall } I = \{1, \dots, n\}, J = \{1, \dots, m\}$$

In this case, the expression first expands to:

$$\sum_{i \in I} (F_{i,1} + F_{i,2} + \dots + F_{i,m}) \quad \text{forall } I = \{1, \dots, n\}$$

Finally, the outer sum is applied to get the expression

$$F_{1,1} + F_{1,2} + \dots + F_{1,m} + F_{2,1} + F_{2,2} + \dots + \dots + F_{n,m}$$

Definition: An *indexed notation* is a formula expressing the fact that the *index operator* is applied to the set of *indexed expressions* constructed by the *index mechanism*., that is, a bijective mapping between the *elements of the index set* and the *indexed expressions*.

4.2.3 Extensions

The general expression of an indexed notation is given by the following notation as seen before:

$$\bigodot_{i \in I} E_i$$

This is really the most general notation, no further concepts are needed to write more complex index notations. This will be shown now by studying several “extensions”, which all can be reduced to this simple form.

4.2.3.1 A list of (indexed) expressions

A list of expressions is normally written as follows:

$$z_i = x_i - y_i \quad \text{forall } i \in I$$

This notation is very common. It says that the equations $z_i = x_i - y_i$ has to be repeated as many times as I has elements. However, this notation could be transformed into the general indexed notation by using the *list operator* as follows:

$$\dot{\bigwedge}_{i \in I} (z_i = x_i - y_i)$$

The index operator is the $\dot{\wedge}$ -operator. The meaning of this operator is to list all indexed expressions. In this case, the indexed expression is normally an equation or an assignment, hence the “side effect” of this operator is to assign a list of values. Of course, in a mathematical text the first notation is preferred, because it is so common. However, this example shows that it can really be seen as an indexed notation. This fact will be important in the context of modeling languages later on, because we can use the same syntax to specify a list of expressions. To express it, we only need to use the list operator instead of any other index operator, as for instance the sigma operator.

4.2.3.2 Index Sets as Expressions

In many situations, the index set is not simply an explicit list of elements, as in $I = \{1, \dots, n\}$. Of course, any set expression can be used to construct the index set I . Hence, I could be the union or the intersection of other sets as in the following expression:

$$I = J \cup K \quad \text{or} \quad I = J \cap K$$

This kind of modification does not change anything for the indexed notation. It is immaterial, how the index set is constructed. Sometimes, however, we construct I as a subset of another set, say J , such that $I \subseteq J$, and we use the notation:

$$I = \{j \in J \mid P(j)\}$$

where $P(j)$ is a property about $j \in J$ that is either true or false. The notation means: “for all $j \in J$ such that $P(j)$ is true”. Clearly, if $P(j)$ is true for all elements in J then the two sets are equal ($I = J$). However, normally $P(j)$ expresses a property that is true only for a proper subset of J . Using I in an indexed notation, we could write:

$$\sum_{i \in I} E_i \quad \text{with} \quad I = \{j \in J \mid P(j)\}$$

However, it is more common and much shorter to use the notation which is as follows:

$$\sum_{j \in J \mid P(j)} E_j$$

Hence, we attach the property $P(j)$ directly to the *indexing term* starting it with the symbol $|$. This is extremely useful and very common in modeling. Suppose we have a list of products. Then it is very common to sum over different subsets of products, for instance “all products for which the demand is larger than x ” or “all products that are green” or whatever. It would be exaggerated to generate an explicit named index set each time.

Here are several examples: Suppose we have $x_1 = 10$, $x_2 = 8$, $x_3 = 2$, $x_4 = 15$, and $x_5 = 22$. Furthermore, $P(i)$ is defined to be “ $x_i \leq 9$ ”, and $Q(i)$ is defined as “ $i \geq 4$ ” with $i \in I = \{1 \dots 5\}$. Then the evaluation of the following expressions is:

$$\sum_{i \in I | P(i)} x_i = x_2 + x_3 = 8 + 2 = 10$$

$$\sum_{i \in I | Q(i)} x_i = x_4 + x_5 = 15 + 22 = 37$$

In the first expression $P(i)$ is true only for x_2 and x_3 , hence the sum take place only over these two terms. In the second expression, $Q(i)$ is true only for $i = 4$ and $i = 5$.

There is no need to explicitly name the properties; their corresponding expressions could be directly used in the indexing terms. Hence the two previous expressions could also be written as:

$$\sum_{i \in I | x_i \leq 9} x_i = x_2 + x_3 = 8 + 2 = 10$$

$$\sum_{i \in I | i \geq 4} x_i = x_4 + x_5 = 15 + 22 = 37$$

We can write any Boolean expression after $| \dots$ in the indexing term. This expression is called *indexing condition*.

Other examples with $k \in K = \{1 \dots 100\}$ are as following:

$$\sum_{k \in K | k^2 \leq 100} k = 1 + 2 + \dots + 10 = 55$$

$$\sum_{k \in K | 3 \leq k \leq 5} k^2 = 3^2 + 4^2 + 5^2 = 50$$

$$\sum_{k \in K | k \text{ is prime}} k = 2 + 3 + 5 + 7 + 11 + 17 + \dots + 97 = 1060$$

In the first expression $k^2 \leq 100$ is only true for $1 \leq k \leq 10$. In the second expression, only 3, 4, 5 are allow for k . In the third, we say to take every k up to 100, such that k is prime (of course, there is no known expression for all primes).

We saw, to extend the indexing term from “ $i \in I$ ” to “ $i \in I | P(i)$ ” does not change the meaning of the indexed notation. The expression $i \in I | P(i)$ is still a set albeit a subset of I that has no explicit name. However, no further concept has been added to the indexed notation.

4.2.3.3 Compound Index Notation

An indexed expression can itself contain an indexed notation, which generates nested indexed notations, as in the following expression:

$$\bigodot_{i \in I} \left(\bigotimes_{j \in J} F_{i,j} \right) \quad (4.5)$$

where \bigodot and \bigotimes are two arbitrary index operators. If the two operators are different then we need to evaluate the inner expression first and the outer afterwards. A notable example is a system of m linear equations with n variables that is commonly noted as follows:

$$\sum_{j \in \{1 \dots n\}} a_{i,j} x_j = b_i \quad \text{with } i \in \{1 \dots m\}$$

Using the *list operator* $\dot{\wedge}$, these formula can be noted as a nested indexed notation, that is, a notation where the index expression contains itself an indexed notation, with the outer index operator $\dot{\wedge}$ and the inner index operator as \sum . The nested notation then is noted as follows:

$$\dot{\wedge}_{i \in \{1 \dots m\}} \left(\sum_{j \in \{1 \dots n\}} a_{i,j} x_j = b_i \right)$$

This notation is a very short writing of the following system of equations:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n-1}x_{n-1} + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n-1}x_{n-1} + a_{2,n}x_n &= b_2 \\ &\dots \\ a_{m,1}x_1 + a_{m,2}x_2 + \dots + a_{m,n-1}x_{n-1} + a_{m,n}x_n &= b_m \end{aligned}$$

On the other hand, if both index operators in (5) are identical, then the two index operators can be merged in the following way:

$$\bigodot_{i \in I} \left(\bigodot_{j \in J} F_{i,j} \right) = \bigodot_{i \in I, j \in J} F_{i,j}$$

It means that we merge the operator and unify also the indexing terms to be written as “ $i \in I, j \in J$ ”. It signifies that we build *every tuple* of the Cartesian Product of the two sets: $(i, j) \in I \times J$. As an example, suppose $I = \{1, 2\}$ and $J = \{a, b, c\}$ then the index set is $I \times J = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)\}$ – it consists of 6 tuples. There are two active indexes i and j , or a tuple (i, j) of active indexes. Hence we also may write this expression in the following way:

$$\bigodot_{(i,j) \in I \times J} F_{i,j}$$

where the index set is $I \times J$ and the active index is replaced by the notation (i, j) which is called an *active index tuple*. In this case, the names within the active index tuples are called *active indices*. The active indices are now place-holders for the single *components* of the tuple.

The index set $I \times J$ is also a set, the set of all tuples in the Cartesian Product. Hence if we do not need to access the single components within the product, we can replace (i, j) by a single active index, say k , and $I \times J$ can be replaced by a set name, say K . Now we see that the previous formula really reduces to our general indexed notation, which is as follows:

$$\bigodot_{(i,j) \in I \times J} F_{i,j} = \bigodot_{k \in K} E_k$$

Since indexed notation with multiple indexing terms really is reducible to indexed notation with one indexing term, we only need to extend the definition of index sets:

Definition: A *compound index set* is any countable (finite or infinite) collection of elements, where all elements are either indivisible items called *atoms* or *tuples* containing all the same number of components.

Some care has to be taken, if the index-sets are tuple lists.

Sometimes access to the tuple itself *and* to its components is needed. In such cases, one could use a combined syntax as in:

$$\bigodot_{k=(i,j) \in I \times J} (E_k, F_{i,j}) \tag{4.6}$$

where $(E_k, F_{i,j})$ is the indexed expression.

Tuples list as index-sets are very common. Let the set I be a number of locations of clients in a region. Then $J \subset I \times I$ is a tuple list and a tuple can be

interpreted as direct connection (route) from a location i to a location j , and let $P_{i,j}$ be true if there is a route from i to j , then the set of all connections is :

$$J = \{(i, j) | P_{i,j}, i, j \in I\}$$

Suppose now, that $f_{i,j}$ is the number of vehicles that are driving from i to j in a hour. Then we can calculate the number of vehicles (x_i) arriving at each location i per hour, it is:

$$x_i = \sum_{j \in I | P_{i,j}} f_{i,j} \quad \text{forall } j \in I$$

How can this be expressed with the tuple list J without using $P_{i,j}$? One variant is:

$$x_i = \sum_{j \in (i, j) \in J} f_{i,j} \quad \text{forall } i \in (i, j) \in J$$

This notation is a little bit clumsy. In this context, $(i, j) \in J$ is often abbreviated as $J[i, j]$ and the expression becomes:

$$x_i = \sum_{j \in J[i, j]} f_{i,j} \quad \text{forall } i \in J[i, j]$$

The previous considerations can be generalized to tuples of more than two components: If I_1, \dots, I_p are $p > 0$ non-empty – not necessary different – index sets containing only atoms, then any subset of the Cartesian product $I_1 \times \dots \times I_p$ is also an index set, called *compound index set*. Its elements are ordered tuples denoted by (i_1, \dots, i_p) with $i_1 \in I_1, \dots, i_p \in I_p$. Each item i_k with $k \in \{1 \dots p\}$ in a tuple (i_1, \dots, i_p) is called a *component* within the tuple. The integer p determines the *arity (dimension) of the tuple*. All elements in a compound index set have the same arity, and the order of the components is significant. If the underlying index sets I_1, \dots, I_p are all ordered, then the order of the corresponding compound index set is determined by the lexicographic ordering: the right most component is varied first.

The *active index tuple* (i_1, \dots, i_p) contains the active indices of each component for the underlying index sets. If an active index is needed for the tuple itself, it can be preceded by a new unique name as in $k = (i_1, \dots, i_p)$, called *named active index tuple*. k is now an active index for the tuple itself. This notation can be extended to any subset of components of the active index tuple. For example, if an active index is used for a sub-tuple consisting of the second, fourth and p -th component besides an index for the tuple itself, one could write this as: “ $k_1 = (i_2, i_4, i_p), k = (i_1, \dots, i_p)$ ”.

We may call this a *list of named active index tuples*. It is important to note that every name for a sub-tuple (the names before the equal signs) as well as all active indices in the *last* (complete) active index tuple must be different from

each other; however, all names in the *other* sub-tuples must occur in the last tuple, because they only define a selection from the last tuple. (Of course, the same selection may turn up more than once, defining several active indices for the same sub-tuple.)

The previous considerations can now be generalized to a list of p indexing terms as follows (where $K \subseteq I_1 \times \dots \times I_p$):

$$\bigodot_{i_1 \in I_1, \dots, i_p \in I_p} E_{i_1, \dots, i_p}, \quad \bigodot_{(i_1, \dots, i_p) \in I_1 \times \dots \times I_p} E_{i_1, \dots, i_p}, \quad \bigodot_{k \in I_1 \times \dots \times I_p} E_k, \quad \bigodot_{k \in K} E_k$$

All tuples in the indexing terms have arity p . All four indexing expressions represent a p -dimensional table of expressions. *Simple index sets* (which contain only atoms) can be interpreted as compound index sets with arity 1. In this case, the parentheses around the "tuple" are not needed.

The use of indexed notation together with *compound index sets*, that is, sets which are subsets of some Cartesian Product, might be considered as a purely theoretical exercise. However, this is not the case. This notation is extremely useful in practice, and it is worth while to understand the mechanism and the notation very well. Therefore, let us give an example.

4.2.4 Exercise 1:

Suppose we have a set of 3 products $P = \{1, 2, 3\}$ and a set of 4 factory locations $F = \{1, 2, 3, 4\}$, where these products are manufactured. The products are transported from one factory to another at unit cost $c_{i,j}$ with $i, j \in F$. Furthermore, each product has a price v_p with $p \in P$. Suppose also, that we transport $x_{p,i,j}$ unities of product p from factory i to factory j with $i, j \in F$ and $p \in P$. The data are as follows (Since x is 3-dimensional, we use the notation $x_{p=1}$ to "slice" (or extract) the matrix $x_{i,j}$ where $p = 1$):

$$v = (6 \ 2 \ 3) \quad c = \begin{pmatrix} - & 1 & 5 & 2 \\ 2 & - & 4 & 2 \\ 1 & 2 & - & 3 \\ 1 & 3 & 1 & - \end{pmatrix} \quad x_{p=1} = \begin{pmatrix} - & 48 & 37 & 45 \\ 36 & - & 43 & 46 \\ 44 & 36 & - & 33 \\ 36 & 39 & 34 & - \end{pmatrix}$$

$$x_{p=2} = \begin{pmatrix} - & 46 & 35 & 39 \\ 32 & - & 47 & 35 \\ 45 & 49 & - & 39 \\ 47 & 46 & 30 & - \end{pmatrix} \quad x_{p=3} = \begin{pmatrix} - & 32 & 32 & 40 \\ 30 & - & 41 & 30 \\ 45 & 43 & - & 45 \\ 44 & 41 & 34 & - \end{pmatrix}$$

Using these data, we now can calculate various data using indexed notation (please verify as an exercise):

(1) The total transportation costs T is calculated as follows:

$$T = \sum_{p \in P, i, j \in F} c_{i,j} \cdot x_{p,i,j} = 3180$$

(2) The total value transported is as follows:

$$V = \sum_{p \in P, i, j \in F} v_p \cdot x_{p,i,j} = 5213$$

(3) The total transportation costs C_p for each product $p \in P$ is:

$$\dot{\bigwedge}_{p \in P} \left(C_p = \sum_{i, j \in F} c_{i,j} \cdot x_{p,i,j} \right) = \begin{pmatrix} 1061 \\ 1096 \\ 1023 \end{pmatrix}$$

(4) The value $W_{i,j}$ transported between all factories $(i, j) \in F \times F$ is:

$$\dot{\bigwedge}_{i, j \in F} \left(W_{i,j} = \sum_{p \in P} v_p \cdot x_{p,i,j} \right) = \begin{pmatrix} - & 476 & 388 & 468 \\ 370 & - & 475 & 436 \\ 489 & 443 & - & 411 \\ 442 & 449 & 366 & - \end{pmatrix}$$

(5) The maximum quantity M_i transported from a factory $i \in F$ is:

$$\dot{\bigwedge}_{i \in F} \left(M_i = \max_{p \in P, j \in F} x_{p,i,j} \right) = (48 \quad 47 \quad 49 \quad 47)$$

(6) For each product $p \in P$ the sum of the maximum quantity S_i transported from a factory $i \in F$ is:

$$\dot{\bigwedge}_{i \in F} \left(S_i = \max_{p \in P} \left(\sum_{j \in F} x_{p,i,j} \right) \right) = \begin{pmatrix} 130 \\ 125 \\ 133 \\ 123 \end{pmatrix}$$

4.2.4.1 Indexed index sets

Index sets occurring in indexing terms can be themselves indexed. (We have already seen an example above in set of sets.) Let us define an index set I and based on it an *indexed index set* $J = \{J_i\}$ where J_i is an index set and $i \in I$. The elements of J are index sets themselves. In this case the index mechanism is

not applied to a singleton but to a set. As an example, suppose $I = \{a, b\}$, $J_a = \{1, 2\}$, and $J_b = \{2, 3\}$ then $J = \{\{1, 2\}, \{2, 3\}\}$. Based on these two sets I and J , one can form the following indexed notation:

$$\bigodot_{i \in I} \left(\bigodot_{j \in J_i} E_{i,j} \right) \quad \text{or} \quad \bigodot_{i \in I, j \in J_i} E_{i,j} \quad (4.7)$$

It is important to note, however, that the index i in $i \in I$ is an active index, whereas the i in J_i is a passive index. Now (7) can be interpreted as follows:

$$\bigodot_{(i,j) \in K} E_{i,j} \quad \text{with } K \subseteq I \times (J_1 \cup \dots \cup J_{|I|}) \quad (4.8)$$

This means that the concept of indexed index set can be reduced to compound index set and does not add any new features. Nevertheless, a notation like (7) may sometimes be more convenient, because the tuples are presented as a hierarchical structure instead of a flat list.

4.2.4.2 Ordering

We saw that in various context an index set has a natural order. If the set is a sequence of time periods or time points, for instance, then we order the elements within this set according to its natural sequence and we expect that the index mechanism is applied in this order. We take an example: Suppose P is a set of time periods or time points. Furthermore, x_p is the production level in period $p \in P$, d_p is the demand in period p , and s_p is the stock level at the end of period p . Then we can build a balance equation at each period:

$$\begin{aligned} & \text{production-level-during-}p \text{ minus demand-during-}p \\ &= \text{stock-at-the-end-of-}p \text{ minus stock-at-the- beginning-of-}p \end{aligned}$$

Formally this can be written as follows (strictly speaking p has two different meaning here: once for a duration (x_p , d_p) and once for a time point (s_p), be careful to model this correctly):

$$x_p - d_p = s_p - s_{p-1} \quad \text{with } p \in P$$

In this example we make a reference to s_{p-1} , that is, to the time period before p . This would have no meaning, if the set P were not ordered. However, if we make reference to a different element then we must be careful to check

of whether the element exists. In the expression above there *is* an undefined reference: if $p = 1$ then the expression $s_{p-1} = s_0$ is not defined. Hence strictly speaking, the expression above is only valid for $p \in P - \{1\}$, we must exclude the first period and the balance equation of the first period is treated apart, or we must define s_0 separately.

4.3. Index Notation in LPL

LPL is a computer language that implements basically the index mechanism described. In this section, this implementation is presented. For a general reference of the LPL language see the Reference Manual (see [72]). LPL implements the indexed notation as close as possible to the mathematical notation. But there are some differences. By mean of several examples, we explain the relationship.

In an LPL code, one needs to specify the index sets first. They are available throughout the whole model code. This is done by a declaration that begins with a keyword **set**. It is followed by the name of the set and an assignment of the elements as follows:

```
set I := 1..10;
set J := 1..100;
set K := 1..n;
parameter n;
```

This declaration defines the three sets $I = \{1 \dots 10\}$, $J = \{1 \dots 100\}$ and a set $K = \{1 \dots n\}$. Note that the identifiers can be declared in any order in LPL. n can or cannot have a value (it must be assigned later on).

This set declarations can be used in expressions. The following expressions:

$$A = \sum_{i \in I} 1 , \quad B = \sum_{j \in J} j , \quad C = \sum_{x \in J | 10 \leq x \leq 20} x^2$$

can be implemented directly as follows:

```
parameter A := sum{i in I} 1;
parameter B := sum{j in J} j;
parameter C := sum{x in J | x >= 10 and x <= 20} x^2;
```

The index operator \sum is the keyword **sum**. The indexing term $i \in I$ is appended to the index operator and bracketed with { and }. We also need to declare new parameters A , B , and C that hold the calculated value. The indexing condition is a proper Boolean expression in the language. Hence, $10 \leq x \leq 20$ can be formulated as $10 \leq x \leq 20$. Note that we use $:=$ as an assignment operator.

If we use tables such as v_i , $a_{i,j}$, and $b_{i,j,k}$ with $i \in I$, $j \in J$, $k \in K$, we must declare them first in the code as follows:

```
parameter v{i in I};
parameter a{i in I, j in J};
parameter b{i in I, j in J, k in K};
```

Using these declaration we now can build other expressions like:

$$D = \sum_{i \in I} v_i, \quad E = \max_{i \in I} \left(\sum_{j \in J} a_{i,j} \right), \quad F_{k \in K} = \sum_{i \in I, j \in J} b_{i,j,k}$$

using the following syntax:

```
parameter D := sum{i in I} v[i];
parameter E := max{i in I} (sum{j in J} a[i,j]);
parameter F{k in K} := sum{i in I, j in J} b[i,j,k];
```

The passive indexes in the indexed expression are enclosed in the brackets [and]. So there is a clear syntactical distinction between the passive and the active indexes in the language.

In LPL there exist a slightly shorter syntax that could also be used for an indexing term. The examples above could be coded as follows:

```
set i := 1..10;
set j := 1..100;
parameter n;
set k := 1..n;

parameter A := sum{i} 1;
parameter B := sum{j} j;
parameter C := sum{j|10<=j<=20} j^2;

parameter v{i};
parameter a{i,j};
parameter b{i,j,k};

parameter D := sum{i} v[i];
parameter E := max{i} (sum{j} a[i,j]);
parameter F{k} := sum{i,j} b[i,j,k];
```

In this code, the active index has *the same name* as the corresponding index set. Hence, {i in I}, an indexing term, can just be written as {i}. This makes the expressions more readable and – in most context – much shorter. However, this can lead to difficulties as soon as the same index set is used more than once in an indexed expression as in the following simple expression:

$$H = \sum_{i,j \in I} c_{i,j}$$

In such a case, LPL allows one to define one or two alias names for the same index set. We then could code this as following:

```
set i, j := 1..10;
parameter c{i,j};
parameter H := sum{i,j} c[i,j];
```

Of course, one always can use the more “traditional” notation as follows:

```
set I := 1..10;
parameter c{i in I, j in J};
parameter H := sum{i in I, j in I} c[i,j];
```

The alias names – separated by commas – are just other names for the index set and can be used as unique active indexes name in indexed notations. LPL allows at most two aliases for each index sets. If we use more than two, then we always need to use the longer syntax as in the following example:

$$L = \sum_{i,j,k,p \in I} d_{i,j,k,p}$$

in which case we need to code this as:

```
set i, j, k := 1..10;
parameter d{i,j,k,p in i};
parameter L := sum{i,j,k,p in i} d[i,j,k,p];
```

The two versions of the LPL code can be downloaded or executed directly at: [exercise0a](#) and [exercise0b](#)

Let us now code the **Exercise 1** given in section [4.2.4](#) as an LPL model. First we implement the declarations: The set of products and factories which are $P = \{1, 2, 3\}$ and $F = \{1, 2, 3, 4\}$ the cost table $c_{i,j}$, the value table v_p , and the quantity table $x_{p,i,j}$ with $i, j \in F$ and $p \in P$. In LPL these data tables can be implemented as follows:

```
set p := 1..3;
set i, j := 1..4;
parameter c{i,j|i<>j} := [. 1 5 2 , 2 . 4 2 , 1 2 . 3 ,
    1 3 1 .];
parameter v{p} := [6 2 3];
parameter x{p,i,j|i<>j} := [
    . 48 37 45 36 . 43 46 44 36 . 33 36 39 34 . ,
    . 46 35 39 32 . 47 35 45 49 . 39 47 46 30 . ,
    . 32 32 40 30 . 41 30 45 43 . 45 44 41 34 . ];
```

In table c and x we use the indexing condition $i <> j$ because there is no cost or quantity defined from a factory to itself. The data itself are listed in a lexicographical order. The expressions given in section 4.2.4 are then formulated as follows in LPL:

```

parameter T      := sum{p,i,j} c[i,j]*x[p,i,j];
parameter V      := sum{p,i,j} v[p]*x[p,i,j];
parameter C{p}   := sum{i,j} c[i,j]*x[p,i,j];
parameter W{i,j} := sum{p} v[p]*x[p,i,j];
parameter M{i}   := max{p,j} x[p,i,j];
parameter S{i}   := max{p} (sum{j} x[p,i,j]);

```

The complete model can be executed at: [exercise1a](#).

Exercise 1 (continued):

Let us now add a requirement that the transportation can only take place between a small subset of all (i, j) links between the factories $i, j \in F$. Hence, we need to introduce a property $K(i, j)$ which is true if the transportation can take place between factory i and factory j . We define it as:

$$K_{i,j \in F} = \{(1, 2) \quad (1, 4) \quad (2, 3) \quad (3, 2) \quad (4, 3)\}$$

The tuple list means that the transportation can only occur from factory 1 to 2, from 1 to 4, from 2 to 3, from 3 to 2 and from 4 to 3. Only 5 transportation links are allowed. The corresponding expressions are then as follows:

(1) The total transportation costs T is calculated as follows:

$$T = \sum_{p \in P, i, j \in F | K(i, j)} c_{i,j} \cdot x_{p,i,j} = 1524$$

(2) The total value transported is as follows:

$$V = \sum_{p \in P, i, j \in F | K(i, j)} v_p \cdot x_{p,i,j} = 2148$$

(3) The total transportation costs C_p for each product $p \in P$ is:

$$C_{p \in P} = \sum_{i, j \in F | K(i, j)} c_{i,j} \cdot x_{p,i,j} = \begin{pmatrix} 511 \\ 537 \\ 476 \end{pmatrix}$$

(4) The value $W_{i,j}$ transported between all factories $(i, j) \in F \times F$ is:

$$W_{i,j \in F \times F} = \sum_{p \in P} v_p \cdot x_{p,i,j} = \begin{pmatrix} - & 476 & 388 & - \\ - & - & 475 & - \\ - & 443 & - & - \\ - & - & 366 & - \end{pmatrix}$$

(5) The maximum quantity M_i transported from a factory $i \in F$ is:

$$M_{i \in F} = \max_{p \in P, j \in F | K(i,j)} x_{p,i,j} = (48 \quad 47 \quad 49 \quad 34)$$

(6) For each product $p \in P$ the sum of the maximum quantity S_i transported from a factory $i \in F$ is:

$$S_{i \in F} = \max_{p \in P} \left(\sum_{j \in F | K(i,j)} x_{p,i,j} \right) = \begin{pmatrix} 85 \\ 47 \\ 49 \\ 34 \end{pmatrix}$$

It is straightforward to code this in LPL. First we declare the table of property $K_{i,j}$ (this is nothing else than a tuple list, see above) as following:

```
| set k{i,j} := [ 1 2 , 1 4 , 2 3 , 3 2 , 4 3 ];
```

The property $K(i,j)$ is simply declared as a set. In fact, in LPL $k\{i,j\}$ defines a tuple list which is a subset of the Cartesian Product of $(i,j) \in I \times I$. In LPL, we can code the expressions as follows:

```
parameter T      := sum{p,i,j|k[i,j]} c[i,j]*x[p,i,j];
parameter V      := sum{p,i,j|k[i,j]} v[p]*x[p,i,j];
parameter C{p}   := sum{i,j|k[i,j]} c[i,j]*x[p,i,j];
parameter W{i,j|k[i,j]} := sum{p} v[p]*x[p,i,j];
parameter M{i}   := max{p,j|k[i,j]} x[p,i,j];
parameter S{i}   := max{p} (sum{j|k[i,j]} x[p,i,j]);
```

Since in LPL $k\{i,j\}$ is itself a set, it can be used as an index set. Hence, the previous code could be written also as:

```
parameter T      := sum{p,k[i,j]} c[i,j]*x[p,i,j];
parameter V      := sum{p,k[i,j]} v[p]*x[p,i,j];
parameter C{p}   := sum{k[i,j]} c[i,j]*x[p,i,j];
parameter W{k[i,j]} := sum{p} v[p]*x[p,i,j];
parameter M{i}   := max{p,j|k[i,j]} x[p,i,j];
parameter S{i}   := max{p} (sum{j|k[i,j]} x[p,i,j]);
```

The difference is in the first four expressions. We wrote $\{p,k[i,j]\}$ instead of $\{p,i,j|k[i,j]\}$. These two indexing terms define the same set. Computationally, however, there can be a huge difference. While the first indexing term $\{p,i,j|k[i,j]\}$ has cardinality of $|P \times I \times I|$ the second indexing term $\{p,k[i,j]\}$ only has cardinality of $|P \times K(i,j)|$. The two version of LPL code can be downloaded or executed directly at: [exercise1b](#) and [exercise1c](#).

4.4. A Complete Problem: Exercise 2

In this second exercise we are going to explore the cost of a given production plan. Suppose we want to manufacture 7 different products. We have 4 identical factories where 5 different machines are installed. 20 different raw products are used to assemble a product. Hence, we have a set of products $P = \{A, B, C, D, E, F, G\}$, a set of factories $F = \{F1, F2, F3, F4\}$, a set of machines $M = \{M1, M2, M3, M4, M5\}$, and a set of raw materials $R = \{1 \dots 20\}$. Additionally, various tables are given with $p \in P$, $f \in F$, $m \in M$, $r \in R$: A table $q_{p,m}$ defines which product can be manufactured on which machine. The table $R_{r,p}$ says how many units of raw materials r are used to manufacture a unit of product p . The table $H_{p,m|q(p,m)}$ says how many hours of machine m it takes to manufacture one unit of product p . The table Rc_r gives the cost of a unit of raw materials, the table Mc_m gives the cost of machine m per hour, and the table S_p is the selling price of product p . Finally, the table $Q_{f,p,m|q(p,m)}$ is a given production plan. It says how many units products to manufacture on which machine and in which factory. (We do not discuss here whether this production plan is good or not.) We suppose that all products can be sold, and that a machine can only process one product at the same time. We do not consider any other costs. A concrete data set is given in the LPL code and at the end of this paper (see [exercise2](#)).

We now wanted to calculate various amounts:

(1) The total selling value TS is:

$$TS = \sum_{f \in F, p \in P, m \in M | q(p,m)} S_p \cdot Q_{f,p,m} = 383646$$

(2) The total cost of raw material RC is:

$$RC = \sum_{f \in F, p \in P, m \in M, r \in R | q(p,m)} R_{r,p} \cdot Rc_r \cdot Q_{f,p,m} = 305882$$

(3) The total machine costs MC is:

$$MC = \sum_{f \in F, p \in P, m \in M | q(p,m)} H_{p,m} \cdot Mc_m \cdot Q_{f,p,m} = 24882$$

(4) The total profit TP is:

$$TP = TS - RC - MC = 52882$$

(5) The total quantity TQ_p produced for each product p is:

$$TQ_p = \sum_{f \in F, m \in M | q_{p,m}} Q_{f,p,m} = (188 \quad 228 \quad 175 \quad 135 \quad 89 \quad 261 \quad 391)$$

(6a) How long (in hours) does the production take on each machine $m \in M$ and in each factory $f \in F$?

$$Ti_{f \in F, m \in M} = \sum_{p \in P | q(p,m)} H_{p,m} \cdot Q_{f,p,m} = \begin{pmatrix} 515 & 312 & 542 & 290 & 330 \\ 548 & 224 & 554 & 324 & 273 \\ 416 & 392 & 309 & 154 & 246 \\ 472 & 216 & 695 & 154 & 452 \end{pmatrix}$$

(6b) How long (in hours) does the production take maximally until the last product leaves a factory $f \in F$?

$$\Pi_{f \in F} = \max_{m \in M} \left(\sum_{p \in P | q(p,m)} H_{p,m} \cdot Q_{f,p,m} \right) = (542 \quad 554 \quad 416 \quad 695)$$

(7a) How big is the loss (gain) of each product ($Lo_{p \in P}$)?

$$Lo_{p \in P} = \sum_{f,m} S_p \cdot Q_{f,p,m} - \sum_{f,m} \left(\sum_r R_{r,p} \cdot Rc_r + H_{p,m} \cdot Mc_m \right) \cdot Q_{f,p,m} \\ = \{55792 \quad 33450 \quad -6320 \quad -19305 \quad 10769 \quad -3046 \quad -18458\}$$

(7b) Which products generate a loss (set $LO_{p \in P}$)?

$$LO_{p \in P} = \sum_{f,m} S_p \cdot Q_{f,p,m} - \sum_{f,m} \left(\sum_r R_{r,p} \cdot Rc_r + H_{p,m} \cdot Mc_m \right) \cdot Q_{f,p,m} < 0 \\ = \{C \quad D \quad F \quad G\}$$

(8) Which machines in which factories work more than 500 hours (set $MO_{f \in F, m \in M}$)?

$$MO_{f,m} = \sum_p H_{p,m} \cdot Q_{f,p,m} > 500 \\ = \{(F1, M1) \quad (F1, M3) \quad (F2, M1) \quad (F2, M3) \quad (F4, M3)\}$$

(9) Which machine and in which factory works more than any other machine
(set $MA_{f \in F, m \in M}$)?²

$$MA_{f,m} =$$

$$f = \underset{f}{\operatorname{argmax}} \left(\sum_p H_{p,m} \cdot Q_{f,p,m} \right) \wedge m = \underset{m}{\operatorname{argmax}} \left(\sum_p H_{p,m} \cdot Q_{f,p,m} \right) \\ = \{(F4, M3)\}$$

(10) Which machine and in which factory has the least work to do (in hours)
(set $MI_{f \in F, m \in M}$)?

$$MI_{f,m} =$$

$$f = \underset{f}{\operatorname{argmin}} \left(\sum_p H_{p,m} \cdot Q_{f,p,m} \right) \wedge m = \underset{m}{\operatorname{argmin}} \left(\sum_p H_{p,m} \cdot Q_{f,p,m} \right) \\ = \{(F3, M4)\}$$

It is straightforward to implement this examples in the language LPL. First we declare the given index sets and the tables as follows (the concrete data for the parameter tables can be found in the LPL model):

```

set
  p := [A B C D E F G];      r := [1..20];
  m := [M1 M2 M3 M4 M5];    f := [F1 F2 F3 F4];
  q{p,m};

parameter
  Q{f,q};      S{p};      R{r,p};      H{q};      Rc{r};      Mc{m}
  };

```

Based on these given data, we can then calculate the various expressions as follows³:

```

parameter
  TS := sum{f, q[p,m]} Q*S;
  RC := sum{f, p, m, r} R*Rc * Q;
  MC := sum{f, p, m} H*Mc * Q;
  TP := TS - RC - MC;

```

²The operators `argmax` and `argmin` are two other index operators returning the element for which the indexed expression has the largest and the smallest value.

³In LPL, it is quite common to leave out the passive indexes: this is very practical and shortens the expressions. However, this practice should only be used by advanced users.

```

Ti{f,m} := sum{p} H*Q;
TI{f} := max{m} sum{p} Mc*Q;
Lo{p} := sum{f,m} S*Q - sum{f,m} (sum{r} R*Rc + H*Mc) *
          Q;
set
  LO{p} := sum{f,m} S*Q - sum{f,m} (sum{r} R*Rc + H*Mc)
          * Q < 0;
  MO{f,m} := sum{p} H*Q > 300;
  MA{f,m} := f=argmax{f} (sum{p} Mc*Q) and m=argmax{m} (
    sum{p} Mc*Q);
  MI{f,m} := f=argmin{f} (sum{p} Mc*Q) and m=argmin{m} (
    sum{p} Mc*Q);

```

4.5. Data for Exercise 2

The data of Exercise 2 are as follows:

$$q^4 = \begin{pmatrix} - & - & 1 & - & 1 \\ 1 & - & 1 & - & - \\ 1 & - & - & 1 & - \\ - & - & - & - & 1 \\ 1 & - & - & - & - \\ - & 1 & 1 & - & - \\ 1 & - & 1 & 1 & - \end{pmatrix} \quad H = \begin{pmatrix} - & - & 8 & - & 8 \\ 4 & - & 7 & - & - \\ 7 & - & - & 2 & - \\ - & - & - & - & 3 \\ 5 & - & - & - & - \\ - & 8 & 2 & - & - \\ 3 & - & 2 & 2 & - \end{pmatrix} \quad S = \begin{pmatrix} 491 \\ 391 \\ 156 \\ 120 \\ 221 \\ 257 \\ 184 \end{pmatrix}$$

⁴ "1" in the matrix means that the combination is possible. For example $q_{B,M1} = 1$ means that product B can be manufactured on machine M1.

$$R = \begin{pmatrix} - & - & - & - & - & - & - & 1 \\ - & - & - & - & - & - & - & - \\ - & 8 & 7 & 6 & - & - & - & - \\ - & 6 & - & - & 5 & - & - & - \\ 7 & 8 & - & - & - & - & - & 5 \\ - & 4 & - & 6 & - & - & - & - \\ 7 & - & - & 3 & 4 & - & - & 4 \\ 5 & - & - & 8 & - & 7 & - & - \\ - & - & - & - & - & 8 & 2 & - \\ - & - & 3 & - & - & - & - & - \\ - & - & - & - & - & - & 4 & - \\ 5 & - & 6 & 4 & - & 7 & 6 & - \\ - & 2 & 5 & - & - & - & 8 & - \\ - & - & 4 & - & - & - & - & 5 \\ - & - & 5 & 3 & - & - & - & - \\ - & 1 & - & - & - & - & - & - \\ - & 5 & - & 1 & - & 6 & 6 & - \\ - & - & 5 & 4 & 8 & 6 & - & - \\ - & 4 & - & - & 3 & - & - & - \\ 4 & - & - & 8 & - & 5 & 3 & - \end{pmatrix}$$

$$Rc = \begin{pmatrix} 5 \\ 6 \\ 7 \\ 3 \\ 7 \\ 7 \\ 3 \\ 5 \\ 8 \\ 4 \\ 4 \\ 8 \\ 4 \\ 2 \\ 3 \\ 8 \\ 5 \\ 6 \\ 5 \\ 6 \end{pmatrix}$$

$$Mc = \begin{pmatrix} 2 \\ 3 \\ 5 \\ 2 \\ 4 \end{pmatrix}$$

$$Q_{f=F1} = \begin{pmatrix} - & - & 15 & - & 27 \\ 21 & - & 40 & - & - \\ 19 & - & - & 16 & - \\ - & - & - & - & 38 \\ 38 & - & - & - & - \\ - & 39 & 40 & - & - \\ 36 & - & 31 & 43 & - \end{pmatrix}$$

$$Q_{f=F2} = \begin{pmatrix} - & - & 15 & - & 18 \\ 11 & - & 48 & - & - \\ 41 & - & - & 18 & - \\ - & - & - & - & 43 \\ 17 & - & - & - & - \\ - & 28 & 32 & - & - \\ 44 & - & 17 & 48 & - \end{pmatrix}$$

$$Q_{f=F3} = \begin{pmatrix} - & - & 10 & - & 24 \\ 16 & - & 15 & - & - \\ 24 & - & - & 20 & - \\ - & - & - & - & 18 \\ 11 & - & - & - & - \\ - & 49 & 31 & - & - \\ 43 & - & 31 & 19 & - \end{pmatrix}$$

$$Q_{f=F4} = \begin{pmatrix} - & - & 36 & - & 43 \\ 34 & - & 43 & - & - \\ 23 & - & - & 14 & - \\ - & - & - & - & 36 \\ 23 & - & - & - & - \\ - & 27 & 15 & - & - \\ 20 & - & 38 & 21 & - \end{pmatrix}$$

4.6. An Optimizing Problem Version

In the previous Exercise 2, several tables have been given to specify a concrete production plan. They are as follows:

1. Given 7 different products ($p \in P$)
2. Manufactured in 4 identical factories ($f \in F$)
3. By means of 5 identical machines in each factory ($m \in M$)
4. Using 20 different raw products ($r \in R$)
5. A table $q_{p,m}$ defining which product p can be manufactured on which machine m
6. A table $R_{r,p}$ saying how many units of raw materials r are used to manufacture a unit of product p
7. A table $H_{p,m|q(p,m)}$ defining the hours of machine m that is taken to manufacture one unit of product p
8. A table Rc_r giving the cost of a unit of raw materials r
9. A table Mc_m specifying the cost of machine m per hour of work
10. A table S_p giving the selling price of product p
11. Finally, a table $Q_{f,p,m|q_{p,m}}$ saying how many units of products p are manufactured in factory f by means of machine m .

Especially, the last table Q is a given table that specifies the *production plan*. It is a given arbitrary plan without asking the question whether this plan is *optimal* in any sense. Let us now ask the question whether this plan could be changed without changing the manufactured quantity of each product, just by switching the production quantity from one machine to another.

4.6.1 Version 1: The Easy Problem

Let's take a concrete example: The present production plan says to manufacture 40 units of the second product (B) in factory F1 with machine M3, that is, we have: $Q_{F1,B,M3} = 40$. These 40 units could also have been manufactured by machine M1, such that $Q_{F1,B,M3}$ becomes 0 and $Q_{F1,B,M1}$ becomes 61. Doing this, implies that the total machine costs MC would now be reduced to:

$$MC = \sum_{f \in F, p \in P, m \in M | q(p,m)} H_{p,m} \cdot Mc_r \cdot Q_{f,p,m} = 23802$$

instead of 24882, which is a cost reduction of 1080 units. We could use probably other reassignment of the production plan in order to reduce the cost even further. However, we would like to do this not on an experimental level, but on a systematic way. Hence, the question is: How should the production plan be, in order to minimize the total machine costs? (Note that the raw material costs do not change, supposing that the quantities of products manufactured do not change.) Expressed in another way: how should the production plan, that is the matrix $Q_{f,p,m}$, be assigned in order to make the machine costs as small as possible.

At this point, things become interesting: From the mathematical point of view, we now have a model in which some data are unknown, that is the matrix $Q_{f,p,m}$ becomes an unknown, because we want to know how this matrix has to be filled to attain our goal (the minimizing of costs). Our model is as follows:

1. Let us introduce a new variable (that is the unknown matrix) to represent the new production plan as follows: $QQ_{f,p,m|q_{p,m}} \geq 0$, where $QQ_{f,p,m}$ is the (unknown) quantity of product p to be manufactured at factory f by the means of the machine m
2. We would like to minimize the total machine costs
3. We also would like to produce the same quantity of each product as in the previous plan

These three conditions could be translated into the following mathematical model. This model has to be solved to find the minimizing cost production plan. The model is as follows:

$$\min \sum_{f \in F, p \in P, m \in M | q(p, m)} H_{p,m} \cdot M c_r \cdot QQ_{f,p,m}$$

subject to

$$\sum_{f \in F, m \in M | q_{p,m}} QQ_{f,p,m} = T Q_p , \quad \text{forall } p \in P$$

The formulation of this model in LPL is as following (see [exercise2a](#) for a complete code):

```

...
integer variable QQ{f,q};
constraint CA{p}: sum{f,m} QQ = sum{f,m} Q;
minimize obj1: sum{f,p,m} H*Mc * QQ;
...

```

The new assigned (optimal, that is “cost minimal”) production plan QQ is as follows:

$$QQ_{f=F1} = \begin{pmatrix} - & - & - & - & 188 \\ 228 & - & - & - & - \\ - & - & - & 175 & - \\ - & - & - & - & 135 \\ 89 & - & - & - & - \\ - & - & 261 & - & - \\ 391 & - & - & - & - \end{pmatrix} \quad QQ_{f=F2} = QQ_{f=F3} = QQ_{f=F4} = 0$$

And the total machine costs are:

$$MC = \sum_{f \in F, p \in P, m \in M | q_{p,m}} H_{p,m} \cdot Mc_r \cdot QQ_{f,p,m} = 16006$$

This is a substantial machine cost reduction of more than 35% compared to the initial (somewhat arbitrary) production plan of Exercise 2 (from 24882 units to 16006 units).

However, there is no need to use the mathematical models approach to find this trivial result! We could have found it by a simple thought: “One should assign as much work as possible to the cheapest machine!” For example, machine M4 only costs 4 to manufacture one unit of product C, while machine M1 costs 14, the only alternative to produce C. Furthermore it is not important in which factory the product is manufactured, since they are all identical. Hence, the total quantity 175 of product C should be manufactured by the means of machine M4. The idea is similar for the other products.

To find the cheapest unit cost for each product and the machine that generates this, we can use two indexed expressions as follows:

- (1) Calculate the cheapest amount of machine cost to manufacture a particular product p (Table Cw_p).

$$Cw_{p \in P} = \min_{m | q_{p,m}} H_{p,m} \cdot Mc_m \\ = \{32 \ 8 \ 4 \ 12 \ 10 \ 10 \ 6\}$$

- (2) What is the machine m that manufactures the cheapest product p (Table $CW_{p,m}$)?

$$CW_{p,m} = (m = \operatorname{argmin}_{m1 \in M | q_{p,m1}} H_{p,m1} \cdot Mc_{m1}) \\ = \{(A, M5) \ (B, M1) \ (C, M4) \ (D, M5) \ (E, M1) \ (F, M3) \ (G, M1)\}$$

Nevertheless, from a practical point of view there might be a problem with this solution. It is certainly the cost minimizing production plan, however the occupation time of the 5 machine in factory F1 is now as follows:

$$Ti_{f \in F, m \in M} = \sum_{p \in P | q(p, m)} H_{p, m} \cdot QQ_{f, p, m} = \begin{pmatrix} 2530 & 0 & 522 & 350 & 1909 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Machine M1 in factory F1 is heavily occupied while all machines in the other factories are idle! (Well, we might evenly distribute the production between the four factories by dividing the quantities $QQ_{f, p, m}$ by four. If the division results not in integers, then we shall round up two and round down two.) Nevertheless, some machines are idle and others are heavily occupied.

4.6.2 Version 2: Limit Capacity

At this point the problem is getting interesting if we add capacity requirements. Suppose that all machines work 10 hour per days, 5 days a week and we would like to manufacture all products within 10 weeks. Is this possible? This problem is all but easy to solve. In fact, we might use the same model as before with the additional constraints that no machine should work more than 500 hours and still we want to minimize costs.

This additional constraint can be formulated as follows:

$$\sum_{p \in P} H_{p, q} \cdot QQ_{f, p, m} \leq 500 , \quad \text{forall } m \in M, f \in F$$

The formulation of this model in LPL is as following (see [exercise2b](#) for a complete code):

```
....  
integer variable QQ{f, q};  
constraint CA{p}: sum{f, m} QQ = sum{f, m} Q;  
CB{m, f}: sum{p} H*QQ <= 500;  
minimize obj1: sum{f, p, m} H*Mc * QQ;  
....
```

The new assigned (optimal, that is “cost minimal”) production plan QQ is now as follows:

$$QQ_{f=F1} = \begin{pmatrix} - & - & - & - & 62 \\ 124 & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ 1 & - & - & - & - \end{pmatrix} \quad QQ_{f=F2} = \begin{pmatrix} - & - & - & - & 62 \\ 89 & - & - & - & - \\ - & - & - & - & 175 \\ - & - & - & - & - \\ - & - & - & - & - \\ - & - & 250 & - & - \\ 48 & - & - & - & - \end{pmatrix}$$

$$QQ_{f=F3} = \begin{pmatrix} - & - & - & - & 12 \\ 1 & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & 134 \\ 1 & - & - & - & - \\ - & - & 11 & - & - \\ 163 & - & - & - & - \end{pmatrix} \quad QQ_{f=F4} = \begin{pmatrix} - & - & - & - & 52 \\ 14 & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ 88 & - & - & - & - \\ - & - & - & - & - \\ - & - & 179 & - & - \end{pmatrix}$$

The following table give the occupation of each machine with this production plan:

$$Ti_{f \in F, m \in M} = \sum_{p \in P | q(p, m)} H_{p, m} \cdot QQ_{f, p, m} = \begin{pmatrix} 499 & 0 & 0 & 0 & 496 \\ 500 & 0 & 500 & 350 & 499 \\ 498 & 0 & 22 & 0 & 498 \\ 496 & 0 & 358 & 0 & 416 \end{pmatrix}$$

We see that no machine exceeds the occupation time of 500 hours. All products will be manufactured within 10 weeks.

Still some machine are not at their capacity limit and some are still idle. Could we reduce the overall production time and to what extend? The most simple way to do this is to experiment with the maximal occupation time. Very quickly we find that we can reduce that time up to 330 hours, which means that the production can be terminated within less than 7 weeks. The corresponding production plan now is as follows:

$$QQ_{f=F1} = \begin{pmatrix} - & - & 33 & - & 41 \\ 3 & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ 63 & - & - & - & - \\ - & 41 & - & - & - \\ 1 & - & 32 & 55 & - \end{pmatrix} \quad QQ_{f=F2} = \begin{pmatrix} - & - & 41 & - & - \\ 80 & - & - & - & - \\ - & - & - & 9 & - \\ - & - & - & - & 110 \\ 2 & - & - & - & - \\ - & 38 & - & - & - \\ - & - & 1 & 50 & - \end{pmatrix}$$

$$QQ_{f=F3} = \begin{pmatrix} - & - & - & - & 38 \\ 80 & - & 10 & - & - \\ - & - & - & - & - \\ - & - & - & - & 6 \\ 2 & - & - & - & - \\ - & 41 & 99 & - & - \\ - & - & 30 & 55 & - \end{pmatrix} \quad QQ_{f=F4} = \begin{pmatrix} - & - & - & - & 34 \\ 54 & - & - & - & - \\ - & - & 165 & - & - \\ - & - & - & - & 18 \\ 22 & - & - & - & - \\ - & 41 & - & - & - \\ - & - & 165 & - & - \end{pmatrix}$$

The following table give the occupation of each machine with this production plan:

$$Ti_{f \in F, m \in M} = \sum_{p \in P | q(p, m)} H_{p, m} \cdot QQ_{f, p, m} = \begin{pmatrix} 330 & 328 & 330 & 330 & 328 \\ 330 & 304 & 330 & 326 & 330 \\ 330 & 328 & 330 & 330 & 330 \\ 330 & 328 & 330 & 330 & 329 \end{pmatrix}$$

As we can see, all machine work almost at 100% of 330 hours. However, the total machine costs are now 21004,4998 units over the minimal costs of 16006, found in the previous model. This is the price we have to pay for a more balanced machine occupation.

4.6.3 Version 3: With Setup Costs

Of course, several important aspects of a real problem have not been considered. For example, we did not take into account setup costs, that is, the costs (or time) to switch the production from one to another product on a particular machine. Suppose that the setup costs to switch production from one product to another on any machine are constant for all machines and all products: 7 units. What is now the optimal production plan?

To model this question we need to “count” the numbers of setups. In the last production plan, for example, we would have to add 3 times a setup cost of 7 for the first machine M1 in first factory F1, etc. However, the number of setups is itself a variable and we cannot count it “afterwards”. The best way is to introduce a “indicator” variable $y_{f, p, m | q_{p, m}} \in \{0, 1\}$ (which is zero or one) for each $QQ_{f, p, m | q_{p, m}}$ quantity that can be produced. Whenever $QQ_{f, p, m | q_{p, m}}$ is strictly greater than zero then $y_{f, p, m | q_{p, m}}$ must be one. This can be formulated as follows:

$$QQ_{f, p, m} > 0 \rightarrow y_{f, p, m} = 1$$

forall $f \in F, p \in P, m \in M$ with $(p, m) \in q_{p, m}$

This is not a mathematical linear function and we need to reformulate it as a proper mathematical constraint in order to use standard general solvers. Fortunately, it is easy to translate this constraint into a linear now as follows (where U is a large number):

$$QQ_{f,p,m} \leq U \cdot y_{f,p,m}$$

forall $f \in F, p \in P, m \in M$ with $(p, m) \in q_{p,m}$

To see that these two constraints express the same condition, we only need to check that y is one if QQ is strictly positive. However, this is always the case, hence the two expressions express the same constraint.

We now have to add all setup costs to the objective function. This function now changed to:

$$\min \sum_{f \in F, p \in P, m \in M | q(p, m)} (H_{p,m} \cdot Mc_r \cdot QQ_{f,p,m} + Setup_p \cdot y_{f,p,m})$$

The formulation of this model in LPL is as following (The value of U has been fixed to 400 and the total machine occupation has slightly been increased to 337):

```

.....
integer variable QQ{f,q};
binary variable y{f,q};
constraint CA{p}: sum{f,m} QQ = sum{f,m} Q;
                CB{m,f}: sum{p} H*QQ <= 337;
                CC{f,q}: QQ <= 400*y;
minimize obj1: sum{f,p,m} H*Mc * QQ + Setup*y;
.....

```

The solution can be look up by running the model [exercise2c](#).

4.7. Conclusion

This paper gave a comprehensive overview of the indexed notation in mathematical modeling. The general notation is defined and several “extensions” have been explained. It is fundamental to understand, read and write mathematical models. We then gave some examples and how they are implemented in the modeling language LPL. Finally, a complete production model with several variants have been given in order to exercise the mathematical notation on a concrete example.

VARIOUS MODELING TOOLS

“A bad workman always blames his tools.”

Virtually hundreds of tools and applications exist to help user to build and to solve mathematical model. Lists of software and applications can be found in the Internet. In this paper, some links are given where to look for an appropriate tool. It is unrealistic to give an exhaustive overview, also because the situation is changing rapidly and new approaches emerge all the time.

The focus in this paper is mathematical *modeling* (not *solving*) in the realm of numerical linear and non-linear optimization and combinatorial models – which are typically used in operations research. The goal of this paper is to give a first impression of various modeling tools –free and commercial– that could be used to build mathematical models. The presentation for each tool is done by a small problem example which is implemented. In this way, the reader gets a first and better idea of the tool. The selection of the tools do not follow a systematical method, it is rather a subjective choice of systems that I came across in the last years.

I also implemented each problem in my modeling language LPL, that I have developed over the years, and I make some comparisons. However, there is no claim here to rate one tool against another. I have my opinion on various features of a modeling tool and I present also my checklist of what a modeling tool should contain and what not, a subjective checklist that comes from my longtime experiences in practical modeling and the implementations. I do not pretend that this checklist is the last resort. There are various criteria why a specific tool is chosen in a concrete situation and not another.

I think *mathematical modeling*, that is, finding and implementing an appropriate formulation of a problem is itself an important activity, besides of *solving* these models, that is, finding efficient methods and algorithms to get an (optimal) solution. A great deal of my research career I invested in developing ideas and implementing them in a modeling language to formulate all kind of problems. Most ideas come from my practical activities as a consulter and teacher in operations research. Modestly, I must say that I was only half-successful: While LPL is a great tool for implementing large MIP's in a commercial context, most of the additional features in LPL are still experimental. Nevertheless, I like to share my knowledge with others, so maybe some of you can pick an idea from my experiences.

The toolboxes and software presented here are grouped into three categories: (algebraic) modeling languages, programming languages, and further tools in the proximity of *mathematical modeling*.

A ZIP file of all models and the data displayed in this paper can be found at this link: [modeling4.7z](#).

5.1. Introduction

The list of mathematical modeling software is huge. One can find free and commercial software for all kind of problems. Searching the Internet under “mathematical modeling”, “mathematical software”, “optimization software”, “mathematics learning”, “mathematical tools” exposes many links. Wikipedia and other sites list various categories, many other links exist, the field is very dynamic and new approaches appear all the time:

- [Modeling Languages](#) lists modeling languages not only mathematical once.
- [What is math. modeling?](#) gives a definition and also presents a serie of videos on mathematical modeling.
- www.mathscareers.org.uk/ is another page that give some explanation what modeling is.
- [Optimization Software](#) list a number of mathematical optimization software (unfortunately some newer are missing).
- [Wikipedia](#)'s list of optimization software (somewhat outdated) contains a large list of modeling software.
- [Wikipedia](#)'s mathematics software contains further links.
- [Comparison of optimization software](#) gives a limited list where several packages are compared.
- [Optimization Software](#) is a large list of systems composed by Prof. Neumaier.
- [10 great languages](#) for mathematics are presented – a subjective but relevant choice in my opinion.
- [Decision tree for optimization software](#) lists a large number of systems.
- [Coin | OR](#) is an open source project for the operations research community. The mission of the COIN-OR Foundation, Inc., is to create and disseminate knowledge related to all aspects of computational operations research.
- A lot of material and models in many languages can be found on [hakank's Home Page](#).
- A List of optimization packages in Python: Update 2023 is found at [Python 2023](#).

In this paper, I present a limited number of systems, hopefully some of the most relevant once, by implementing a simple example for each tool to get a flavor and a first impression of them. I give my personal comments in the light of the following criteria for a – what I think is a “good” – mathematical modeling system. In my opinion, a mathematical modeling system should fulfill the following requirements:

1. *A modeling system should be based in a formal language.* Like a programming language, which is based on a written text and a formal syntax, a modeling system should be based on an executable language.
2. The modeling language syntax should be as close as possible to a common mathematical notation used to specify a model.
3. In addition, the modeling language should be a complete programming language, that is, rich enough to implement any algorithm (Turing complete), and it should contain features of a modern programming language.
4. An important part of the syntax should be its (sparse) index capability in order to be able to formulate large models in a concise way.
5. It should be possible to formulate within the language all kinds of model paradigms: linear, non-linear, permutations, containing logical constraints, constraint programming (CP), differential systems, etc.
6. In the language it should be possible to modularize a model into sub-modules, in order to encapsulate entities and objects, similar to a modern programming language that contains classes (objects), modules, procedures, functions, etc.
7. The model formulation should be independent of a solution method (solver). From the structure of the model, it should be possible to infer automatically what solver is apt to solve the model.
8. The model should be independent from the data instances. A model structure should be executable without data. The data could be part of the code, but it is not necessary and normally would be separated from the model structure.
9. On the base of the language, visual representations, like A-C graphs or others, and “visual” editors to build, manipulate and to modify the model could be built, as alternatives to editing and viewing the textual code – a modeling framework.
10. Likewise programming languages, documenting a model is an important part in modeling – I guess it is even more important in modeling to understand its semantic.

In my opinion, mathematical modeling should not be just an “addon” package within a common programming language, this would always be somewhat artificial. Modeling is too important to be a supplement or an annex of an (existing) programming language. I know this is not the actual trend, a lot of packages have been developed for Python, Julia, and other languages that allow a modeler to code mathematical models. It is said that this has some advantages: One must learn only one language; a language like Python contains thousands of packages for all kinds of tasks: manipulating, reading/writing data, generating graphics and others; implementing efficient algorithms.

However, the modeling language that I have in mind (which has not been found by now in my opinion) *is also* a complete programming language with all its interfaces to libraries, other software and own extensions with packages. Another paper that compares five languages (AMPL, AIMMS, GAMS, Pyomo, and JuMP) is [81].

5.2. (Algebraic) Modeling Languages

Algebraic Modeling Languages are computer executable specialized language to represent mathematical models. Their syntax is similar to the mathematical notation of optimization problems: They use sets, indices, multi-dimensional data-cube, variables, algebraic expressions to specify constraints and objectives. The model representation is purely declarative and does not give any indication on how to process or solve the model. The language has an interface to one or several separate “solvers” – procedures or libraries that can solve the model. The main task of the language is to create a complete model instance (model structure + data), to call a solver, and then to output an appropriate result. Depending on the model type (linear, integer, non-linear, etc.) the model must be solved by a different solver. Some solvers are limited (but eventually efficient) to solve linear model, others are specialized on non-linear differentiable problems, still others on Boolean problems, or others. Normally, an (existing) algebraic language is not a full-fledged programming language, or it includes a limited number of procedures to manipulate data or to output results.

One of the first algebraic modeling language was GAMS, developed in the 1980's, and is still a major and prominent player on the market. Shortly after, a first version of AMPL, a powerful tool, came out and is one of the most widespread language.

5.2.1 GAMS

GAMS is a general algebraic modeling system, based on its high level modeling language. It declares a model by specifying a number of *Sets*, *Parameters*, *Tables*, *Scalars*, *Variables*, *Equations*, *Model*. It contains a *Solve* statement to call a solver and various *Display* functions to output the results. GAMS has probably the largest number of interfaces to all kinds of solvers. Very large models can be processed efficiently using GAMS and it has a powerful sparse-indexing mechanism. A large number of model examples is available on the site of GAMS. GAMS is probably the oldest algebraic modeling language, its first version has been published in 1982. It is a commercial software, but a free demo version is available. As a small example, the following model implements the problem of maximally coexisting queens on a chessboard.

5.2.1.1 Coexisting Armies of Queens ([coexx](#))

— Run LPL Code , HTML Document –

Problem: Two armies of queens (black and white) peacefully coexist on a chessboard when they are placed on the board in such a way that no two queens from opposing armies can attack each other. The problem is to find the maximum two equal-sized armies. This model is from the GAMS model library (GAMS SEQ=218). See also [57].

Modeling Steps: The model is a tighter formulation of the model [coex](#). The chessboard is a 8×8 grid, defining $i \in I = \{1, \dots, 8\}$ rows and $j \in I$ columns. It contains $s \in S = \{1, \dots, 2|I| - 3\}$ diagonals. Let sh_s be the shift value of a (forward) diagonal relative to the main diagonal, and let $rv_{s,i}$ be the reverse shift order value.

We introduce various binary variables: $xw_{i,j} = 1$ if cell (i, j) has a white queen; $xb_{i,j} = 1$ if cell (i, j) has a black queen; $wa_i = 1$ if row i contains a white queen; $wb_j = 1$ if column j contains a white queen; $wc_s = 1$ if (forward) diagonal s contains a white queen; $wd_s = 1$ if (backward) diagonal s contains a white queen. Furthermore, let us introduce the total number of white (black) queens as an integer variable tot.

Constraints (1-6) are formulated in a logical way. Basically, these constraints say the following: if a cell occupies a white queen then in the same row, column and the two diagonals only white queens are allowed, and if a cell occupies a black queen then no white queen is allowed in the same row, column and the two diagonals. Constraint (1), for instance, formulates the following fact: “if cell (i, j) is occupied by a white queen ($xw_{i,j}$ is true) then row i and column j must contain a white queen”, that is, wa_i and wb_j both must be true. The other logical constraints can be interpreted in a similar way (try to verbalize them yourself!).

Constraints (7-8) define the total number of white and black queens and the objective function maximized that number. (In addition, we fix the position of one queen in the LPL code, since this does not change the optimal value.)

$$\begin{aligned}
 \max \quad & \text{tot} \\
 \text{max } & xw_{i,j} \rightarrow wa_i \wedge wb_j \quad \text{forall } i, j \in I \quad (1) \\
 & xw_{i,i+sh} \rightarrow wc_s \quad \text{forall } s \in S, i \in I \quad (2) \\
 & xw_{i,i+rv_{s,i}} \rightarrow wd_s \quad \text{forall } s \in S, i \in I \quad (3) \\
 \text{max } & xb_{i,j} \rightarrow \neg wa_i \wedge \neg wb_j \quad \text{forall } i, j \in I \quad (4) \\
 & xb_{i,i+sh} \rightarrow \neg wc_s \quad \text{forall } s \in S, i \in I \quad (5) \\
 & xb_{i,i+rv_{s,i}} \rightarrow \neg wd_s \quad \text{forall } s \in S, i \in I \quad (6) \\
 \text{max } & \text{tot} = \sum_{i,j} xb_{i,j} \quad (7) \\
 \text{max } & \text{tot} = \sum_{i,j} xw_{i,j} \quad (8)
 \end{aligned}$$

Further Comments: The model is a typical model with logical constraints. LPL translates these constraints automatically into linear integer constraints (for more information see [my Logical Paper](#)). For instance, take the first constraint:

$$xw_{i,j} \rightarrow wa_i \wedge wb_j \quad \text{forall } i, j \in I$$

First it is transformed to a CNF-form (conjunctive normal form) as follows:

$$(\neg xw_{i,j} \vee wa_i) \wedge (\neg xw_{i,j} \vee wb_j) \quad \text{forall } i, j \in I$$

And then to two constraints as follows:

$$1 - xw_{i,j} + wa_i \geq 1 \quad , \quad 1 - xw_{i,j} + wb_j \geq 1 \quad \text{forall } i, j \in I$$

In GAMS, these constraints must be translated manually to mathematical linear constraints. GAMS has this very handy *Alias* syntax to specify several names for one index (In LPL, it can be placed directly in the declaration of the sets). This unimpressive feature is helpful for large and complicated models, no other language contains it. GAMS has a somewhat “old-fashioned” syntax like $=g=$ for \geq . One also needs to declare an equation first, and only after that the constraint expression can be assigned. Comments are not clearly marked or separated from the formal syntax.

Note also, GAMS is – apart from some output statements – a purely declarative language and, therefore, has no instruction to generate graphs.

LPL code (run `coexx`)

```
model COEXX "Coexisting Armies of Queens";
set i, j := 1..8 "size of chess board";
s := 1..2*i-3 "diagonal offset";
parameter sh{s} := s-#i+1 "diag shift values";
rv{s,i} := #i+1-2*i+ sh "reverse shift order";
binary variable
  xw{i,j} "has a white queen";
  xb{i,j} "has a black queen";
  wa{i}   "white in row i";
  wb{i}   "white in column j";
  wc{s}    "white in diagonal s";
  wd{s}    "white in backward diagonal s";
variable tot;
constraint
  aw{i,j}: xw -> wa[i] and wb[j]
    "white in row i/col j";
  cw{s,i}: xw{i,i+sh} -> wc[s]
    "white in diag s";
  dw{s,i}: xw{i,i+rv} -> wd[s]
    "white in back diag s";
  ab{i,j}: xb -> ~wa[i] and ~wb[j]
```

```

    "black in row i/col j";
cb{s,i}: xb[i,i+sh] -> ~wc[s]
    "black in diag s";
db{s,i}: xb[i,i+rv] -> ~wd[s]
    "black in back diag s";
eb: tot = sum{i,j} xb
    "total black";
ew: tot = sum{i,j} xw
    "total white";
fx11: xb[1,1] = 1;
    --fix one queen in the NW corner
maximize obj: tot;
// draw the solution blackboard
Draw.Scale(50,50);
{i,j} Draw.Rect(i,j,1,1,if((i+j)%2,0,1),0);
{i,j|xb} Draw.Circle(j+.5,i+.5,.3,5);
{i,j|xw} Draw.Circle(j+.5,i+.5,.3,4);
end

```

GAMS code (download [coexx.gms](#))

```

Sets i size of chess board      / 1* 8 /
      s diagonal offsets        / 1* 13 /
scalar idiags correct size of s;
idiags = 2*card(i) - 3;
abort$(card(s) <> idiags) 's has incorrects size',idiags;
Alias (i,j)
Parameter sh(s)      shift values for diagonals
              rev(s,i) reverse shift order;
sh(s)  = ord(s) - card(i) + 1 ;
rev(s,i) = card(i) + 1 - 2*ord(i) + sh(s);

Binary Variable
xw(i,j) has a white queen
xb(i,j) has a black queen
wa(i)   white in row i
wb(i)   white in column j
wc(s)   white in diagonal s
wd(s)   white in backward diagonal s;
Variable tot;

Equations aw(i,j) white in row i
            bw(j,i) white in column j
            cw(s,i) white in diagonal s
            dw(s,i) white backward diagonal s
            ew   total white
            ab(i,j) black in row i
            bb(j,i) black in column j
            cb(s,i) black in diagonal s
            db(s,i) black backward diagonal s

```

```

eb      total black;
aw(i,j).. wa(i) =g= xw(i,j);
bw(j,i).. wb(j) =g= xw(i,j);
cw(s,i).. wc(s) =g= xw(i,i+sh(s));
dw(s,i).. wd(s) =g= xw(i,i+rev(s,i));

```

Solution output of the LPL drawing code

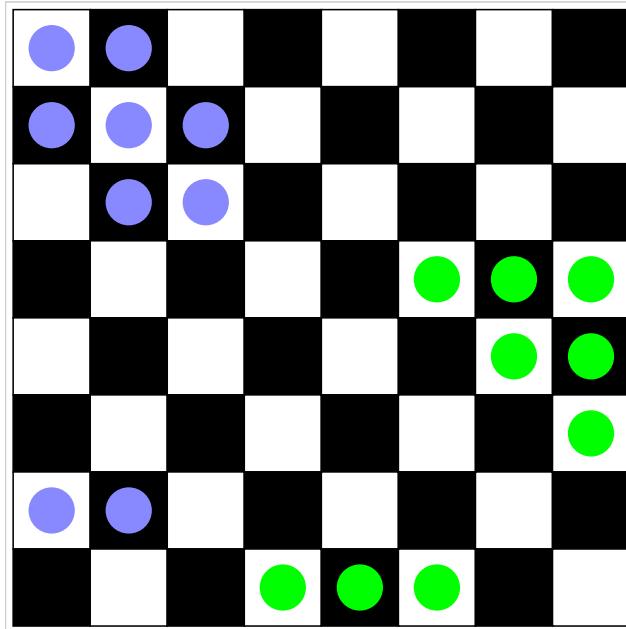


Figure 5.1: 2×9 Queens can be placed maximally

...continue with GAMS code

```

ab(i,j).. 1-wa(i) =g= xb(i,j);
bb(j,i).. 1-wb(j) =g= xb(i,j);
cb(s,i).. 1-wc(s) =g= xb(i,i+sh(s));
db(s,i).. 1-wd(s) =g= xb(i,i+rev(s,i));
eb.. tot =e= sum((i,j), xb(i,j));
ew.. tot =e= sum((i,j), xw(i,j));
Model army / all /;
option limcol=0,limrow=0;
xb.fx('1','1') = 1; ! fix one position in the NW corner
Solve army maximizing tot using mip;

```

5.2.2 AMPL

AMPL is one of the most powerful algebraic modeling language available today. It also supports dozens of solvers. It is similar to GAMS but has a more straightforward and more concise syntax. It can formulate large linear, integer and non-linear models and contains automatic differentiation for non-linear models. It also can add logical constraints that are treated by constraint programming solvers. AMPL generates the **nl**-file, which has become a standard input format for many solvers. Model data are strictly separated from the model structure which is purely declarative. In a small, separated script language one can read/write the data and call a solver or do some other model manipulation interactively. In the following, two small models show the syntax of AMPL.

5.2.2.1 Transshipment Problem with Fixed Costs ([multmip1](#))

— Run LPL Code , [HTML Document](#) —

Problem: The model is a transportation problem from a set of origins to destinations with variable and fixed costs on each route if used. The model is from [19]

Modeling Steps: A set of products $p \in P$ must be transported from origins (p.e. warehouses) $i \in O$ to destinations (p.e. client regions) $j \in D$. The supply quantity at the origin i per product p is given as $s_{i,p}$, the demand at the destination j for product p is $d_{j,p}$. Maximally, $u_{i,j}$ units of the products can be transported on an route (i,j) . Variable transportation costs on route (i,j) for product p is $vc_{i,j,p}$. There is also a fixed cost on route (i,j) if this route is used: $fc_{i,j}$. The unknown transporation quantity is a variable $T_{i,j,p}$. Furthermore, we need a binary variable $U_{i,j}$, which is 1 if the route (i,j) is used. The model is:

$$\begin{aligned} \min \quad & \sum_{i \in O, j \in D, p \in P} vc_{i,j,p} \cdot T_{i,j,p} + \sum_{i \in O, j \in D} fc_{i,j} \cdot U_{i,j} \\ \text{s.t.} \quad & \sum_{j \in D} T_{i,j,p} = s_{i,p} \quad \text{forall } i \in O, p \in P \\ & \sum_{i \in O} T_{i,j,p} = d_{j,p} \quad \text{forall } j \in D, p \in P \\ & \sum_{p \in P} T_{i,j,p} \leq u_{i,j} \cdot U_{i,j} \quad \text{forall } i \in O, j \in D \end{aligned}$$

Further Comments: This model shows how close the AMPL (and LPL) notation is to the mathematical notation. To run the model, AMPL uses a separated script language :

```
model multmip1.mod;
data multmip1.dat;
solve;
option display_1col 5;
display Trans;
```

In LPL, one can also separate data and output instructions from the model structure and then run the following instruction from the command line :

```
lplc multmip1 - @INF='multmip1.txt' #@OUTF='multmip1.out'
```

LPL code (run **multmip1**)

```
model multmip1 "Transshipment Problem with fixed costs";
set ORIG;      // origins
set DEST;      // destinations
set PROD;      // products

parameter supply{ORIG,PROD};
parameter demand{DEST,PROD};
parameter limit{ORIG,DEST};
parameter vcost{ORIG,DEST,PROD};

variable Trans{ORIG,DEST,PROD};
parameter fcost{ORIG,DEST};
binary variable Use{ORIG,DEST};

minimize TotalCost:
  sum{i in ORIG, j in DEST, p in PROD} vcost[i,j,p] *
    Trans[i,j,p]
  + sum{i in ORIG, j in DEST} fcost[i,j]*Use[i,j];
constraint
  Supply{i in ORIG, p in PROD}: sum{j in DEST} Trans[i,
    j,p] = supply[i,p];
  Demand{j in DEST, p in PROD}: sum{i in ORIG} Trans[i,
    j,p] = demand[j,p];
  Multi{i in ORIG, j in DEST} : sum{p in PROD} Trans[i,
    j,p] <= limit[i,j] * Use[i,j];
end
```

LPL output model (download **multmip1.out**)

```
model output;
Write('      %5s\n', {j in DEST} j);
Write{i in ORIG} ('%4s %5d\n', i, {j in DEST} sum{p in
  PROD} Trans[i,j,p]);
end
```

AMPL code (download **multmip1.mod**)

```

set ORIG;      # origins
set DEST;      # destinations
set PROD;      # products

param supply {ORIG,PROD} >= 0;   # amounts available at
    origins
param demand {DEST,PROD} >= 0;   # amounts required at
    destinations

check {p in PROD}:
    sum {i in ORIG} supply[i,p] = sum {j in DEST}
        demand[j,p];

param limit {ORIG,DEST} >= 0;   # maximum shipments on
    routes
param vcost {ORIG,DEST,PROD} >= 0; # variable shipment
    cost on routes
var Trans {ORIG,DEST,PROD} >= 0;   # units to be shipped
param fcost {ORIG,DEST} >= 0;       # fixed cost for using
    a route
var Use {ORIG,DEST} binary;          # = 1 only for routes
    used

minimize Total_Cost:
    sum {i in ORIG, j in DEST, p in PROD} vcost[i,j,p] *
        Trans[i,j,p]
    + sum {i in ORIG, j in DEST} fcost[i,j] * Use[i,j];

subject to Supply {i in ORIG, p in PROD}:
    sum {j in DEST} Trans[i,j,p] = supply[i,p];
subject to Demand {j in DEST, p in PROD}:
    sum {i in ORIG} Trans[i,j,p] = demand[j,p];
subject to Multi {i in ORIG, j in DEST}:
    sum {p in PROD} Trans[i,j,p] <= limit[i,j] * Use[i,j];

```

LPL data ([download multmip1.txt](#))

```

model data;
ORIG := [GARY CLEV PITT] ;
DEST := [FRA DET LAN WIN STL FRE LAF] ;
PROD := [bands coils plate] ;
supply{ORIG,PROD} := /
    : (tr) GARY    CLEV    PITT :
        bands    400     700     800
        coils    800    1600    1800
        plate    200     300     300 /;
demand{DEST,PROD} := /
    : (tr) FRA DET LAN WIN STL FRE LAF :
        bands   300 300 100 75 650 225 250
        coils   500 750 400 250 950 850 500

```

```

    plate 100 100 0 50 200 100 250 /;
limit{ORIG,DEST} := 625 ;
vcost{ORIG,DEST,PROD} := /
[*,* ,bands]:FRA DET LAN WIN STL FRE LAF :
    GARY 30 10 8 10 11 71 6
    CLEV 22 7 10 7 21 82 13
    PITT 19 11 12 10 25 83 15
[*,* ,coils]:FRA DET LAN WIN STL FRE LAF :
    GARY 39 14 11 14 16 82 8
    CLEV 27 9 12 9 26 95 17
    PITT 24 14 17 13 28 99 20
[*,* ,plate]:FRA DET LAN WIN STL FRE LAF :
    GARY 41 15 12 16 17 86 8
    CLEV 29 9 13 9 28 99 18
    PITT 26 14 17 13 31 104 20 /;
fcost{o,d} := /
    : FRA DET LAN WIN STL FRE LAF :
    GARY 3000 1200 1200 1200 2500 3500 2500
    CLEV 2000 1000 1500 1200 2500 3000 2200
    PITT 2000 1200 1500 1500 2500 3500 2200 /;
end

```

AMPL data (download [multmip1.dat](#))

```

data;
set ORIG := GARY CLEV PITT ;
set DEST := FRA DET LAN WIN STL FRE LAF ;
set PROD := bands coils plate ;
param supply (tr): GARY CLEV PITT :=
    bands 400 700 800
    coils 800 1600 1800
    plate 200 300 300 ;
param demand (tr):
    FRA DET LAN WIN STL FRE LAF :=
bands 300 300 100 75 650 225 250
coils 500 750 400 250 950 850 500
plate 100 100 0 50 200 100 250 ;
param limit default 625 ;
param vcost :=
[*,* ,bands]:
    FRA DET LAN WIN STL FRE LAF :=
    GARY 30 10 8 10 11 71 6
    CLEV 22 7 10 7 21 82 13
    PITT 19 11 12 10 25 83 15
[*,* ,coils]:
    FRA DET LAN WIN STL FRE LAF :=
    GARY 39 14 11 14 16 82 8
    CLEV 27 9 12 9 26 95 17
    PITT 24 14 17 13 28 99 20

```

```

[*,*,*plate]:
    FRA DET LAN WIN STL FRE LAF :=
GARY  41   15   12   16   17   86   8
CLEV  29   9    13   9    28   99   18
PITT  26   14   17   13   31   104  20 ;
param fcost:
    FRA DET LAN WIN STL FRE LAF :=
GARY  3000 1200 1200 1200 2500 3500 2500
CLEV  2000 1000 1500 1200 2500 3000 2200
PITT  2000 1200 1500 1500 2500 3500 2200 ;

```

5.2.2.2 Anti-Assignment Problem ([balAssign0](#))

— Run LPL Code , [HTML Document](#) –

Problem: Assign people into groups in a way that the groups are as heterogeneous as possible. Each person is identified by four characteristics (department, location, rating, and title). Build 7 groups such that these characteristics are distributed between the groups as much as possible. Let $typ_{i,k}$ be the characteristic (as string) of person i in category k , and let w_k be the weight in the objective function for category k . Finally, the relation $TYPES_{k,t}$ is true if characteristic t is in category k . This problem is from [AMPL/Logic](#)

Modeling Steps: We define a set of people as $i \in I$ and a set of groups as $j \in J$. All the characteristics $t \in T$ are grouped into four categories $k \in K$.

A binary variable $X_{i,j}$ (assign) is introduced and is 1 if person i is in group j . $MinInGrp$ and $MaxInGrp$ are two variables defining a minimal/maximal number of persons in all groups. Finally, let $minT_{k,t}$ and $maxT_{k,t}$ the minimal/maximal number of persons that have characteristic t in all groups. The model is then :

$$\begin{aligned}
& \text{min} && MaxInGrp - MinInGrp + \sum_{(k,t) \in TYPES} w_k (maxT_{k,t} - minT_{k,t}) \\
& \text{subject to} && \sum_j X_{i,j} = 1 \quad \text{forall } i \in I \\
& && MinInGrp \leq \sum_i X_{i,j} \leq MaxInGrp \quad \text{forall } j \in J \\
& && minT_{k,t} \leq \sum_{i|typ_{i,k}=t} X_{i,j} \leq maxT_{k,t} \quad \text{forall } j \in J, (k,t) \in TYPES
\end{aligned}$$

The objective function minimizes the deviations: Basically, we want the same number of persons in the 7 groups ($MaxInGrp - MinInGrp$), but also the

same number of all characteristics in each category in the same group. The first constraint then states that each person must be in one single group. The other two constraints limit the deviations. To make this problem harder, decrease sample and/or increase the number of groups.

Further Comments: This model shows the difference in set/indices syntax in AMPL and LPL. Index and sets can have the same name in LPL. This may lead to some confusion initially, but it makes the model much shorter and in my opinion more readable. It shows also, that AMPL is more powerful in set definitions (set of sets are possible, for instance and others), but this is outweighed in LPL by its algorithmic (programming) part. In LPL, all “compound sets” are defined as multi-dimensinal “relations”.

LPL code (run **balAssign0**)

```

model balAssign0 "Anti-Assignment Problem";
set PEOPLE,i;
set GROUPS,j;
set CATEG,k "categories";
set t "all characteristics";
string parameter typ{i,k};
parameter typeWt{k};
set TYPES{k,t};

binary variable Assign{i,j};
variable MinInGrp [0..Floor(#i/#j)];
variable MaxInGrp [Ceil(#i/#j-1)..9999];
variable MinType{TYPES[k,t]} [0..Floor((sum{i|typ=t}1
    /#j)];
variable MaxType{TYPES[k,t]} [Ceil(sum{i|typ=t}1/#j)
    ..9999];

minimize Variation: (MaxInGrp-MinInGrp) +
    sum{TYPES[k,t]} typeWt*(MaxType-MinType);
constraint
    AssignAll{i}: sum{j} Assign[i,j] = 1;
    InGrpDefn{j}: MinInGrp <= sum{i} Assign[i,j] <=
        MaxInGrp;
    TypeDefn{j,TYPES[k,t]}:
        MinType <= sum{i|typ=t} Assign[i,j] <= MaxType;
    Write{j}('group %s\n%s\n', j, Format{i|Assign}('%s %11s
        \n',i, {k} typ));
end

```

AMPL code ([download balAssign0.mod](#))

```

set ALL_PEOPLE ordered;
param sample integer > 0;
param selection integer >= 0, < sample;
set PEOPLE := {i in ALL_PEOPLE: ord(i) mod sample =
    selection};

```

```

set CATEG;
param type {ALL_PEOPLE,CATEG} symbolic;
param typeWt {CATEG} >= 0;
param numberGrps integer > 0;
set TYPES{k in CATEG}:=setof{i in PEOPLE} type[i,k];
var Assign {i in PEOPLE, j in 1..numberGrps} binary;
var MinInGrp <= floor (card(PEOPLE)/numberGrps);
var MaxInGrp >= ceil (card(PEOPLE)/numberGrps);
var MinType {k in CATEG, t in TYPES[k]}
    <= floor (card {i in PEOPLE: type[i,k] = t} /
    numberGrps);
var MaxType {k in CATEG, t in TYPES[k]}
    >= ceil (card {i in PEOPLE: type[i,k] = t} /
    numberGrps);
minimize Variation: (MaxInGrp - MinInGrp) +
    sum {k in CATEG, t in TYPES[k]}
        typeWt[k] * (MaxType[k,t] - MinType[k,t]);
subj to AssignAll {i in PEOPLE}:
    sum {j in 1..numberGrps} Assign[i,j] = 1;
subj to MinInGrpDefn {j in 1..numberGrps}:
    MinInGrp <= sum {i in PEOPLE} Assign[i,j];
subj to MaxInGrpDefn {j in 1..numberGrps}:
    MaxInGrp >= sum {i in PEOPLE} Assign[i,j];
subj to MinTypeDefn {j in 1..numberGrps, k in CATEG, t in
    TYPES[k]}:
    MinType[k,t] <= sum {i in PEOPLE: type[i,k] = t}
        Assign[i,j];
subj to MaxTypeDefn {j in 1..numberGrps, k in CATEG, t in
    TYPES[k]}:
    MaxType[k,t] >= sum {i in PEOPLE: type[i,k] = t}
        Assign[i,j];

```

LPL data (download **balAssign0.txt**)

```

model data "TEST DATA for balassign0.lpl";
integer parameter sample [0..9999] := 6;
integer parameter selection [0..sample-1] := 0;
integer parameter numberGrps := 7;;
j:=1..numberGrps;
set ALL_PEOPLE := [
    BIW AJH FWI IGN KWR KKI HMN SML RSR TBR
    KRS CAE MPO CAR PSL BCG DJA AJT JPY HWG
    TLR MRL JDS JAE TEN MKA NMA PAS DLD SCG
    ... cut lines ...
    JCO PSN SCS RDL TMN CGY GMR SER RMS JEN
    DWO REN DGR DET FJT RJZ MBY RSN REZ BLW];;

typeWt{k} := / dept 1 loc 1 rate 1 title 1 /;
string parameter type{ALL_PEOPLE,CATEG} := /
    : dept          loc          rate          title :

```

```

BIW    NNE    Peoria      A    Assistant
KRS    WSW    Springfield  B    Assistant
TLR    NNW    Peoria      B    Adjunct
... cut lines ...
XYF    NNE    Peoria      A    Assistant
JEN    NNE    Peoria      A    Deputy
BLW    NNE    Peoria      A    Deputy /;

{a in ALL_PEOPLE} if(a%sample = selection, Addm(PEOPLE,a
  &'));
{CATEG,ALL_PEOPLE} (Addm(t,type));
{i in ALL_PEOPLE,k} (typ[i within PEOPLE,k]:=type[i,k])
  ;
{i,k,t} if(typ[i,k]=t,TYPES[k,t]:=1);
parameter MinType1{TYPES[k,t]} := Floor(sum{i|typ=t}1/
  numberGrps);
end

```

AMPL data (download [balAssign0.dat](#))

```

param sample := 6;
param selection := 0;
param numberGrps := 7;
set ALL_PEOPLE :=
  BIW AJH FWI IGN KWR KKI HMN SML RSR TBR
  KRS CAE MPO CAR PSL BCG DJA AJT JPY HWG
  TLR MRL JDS JAE TEN MKA NMA PAS DLD SCG
  ... cut lines ...
  JCO PSN SCS RDL TMN CGY GMR SER RMS JEN
  DWO REN DGR DET FJT RJZ MBY RSN REZ BLW;

param: CATEG: typeWt := dept 1  loc 1  rate 1  title 1;
param type:
  dept      loc      rate      title   :=
BIW    NNE    Peoria      A    Assistant
KRS    WSW    Springfield  B    Assistant
TLR    NNW    Peoria      B    Adjunct
... cut lines ...
XYF    NNE    Peoria      A    Assistant
JEN    NNE    Peoria      A    Deputy
BLW    NNE    Peoria      A    Deputy ;

```

5.2.3 LINDO/LINGO

LINDO Systems is a software for integer programming, linear programming, nonlinear programming, stochastic programming, global optimization. It contains a modeling language (LINGO) as well as solvers integrated. The *What's*

Best library – based on this system – is a well known addon for Excel. It contains a fully featured environment for building and editing problems with LINGO language. A package to Python exists: the pyLingo package. The LINGO language contains also the different parts: SETS:...ENDSETS for defining sets, DATA:...ENDDATA for defining data tables, and For-loops for defining sets of constraints. However, the LINGO language is not as powerful as GAMS or AMPL. Also its syntax is a bit clumsy. It uses sets, but then needs for-loops. Some keywords are old style (@FOR).

In the following, a model for the “Data Envelopment Analysis” field is used to show the syntax of LINGO.

5.2.3.1 Data Envelopment Analysis (**deamod**)

— Run LPL Code , HTML Document –

Problem: Data Envelopment Analysis (DEA) is a method in economics to empirically measure, evaluate and compare the efficiency of decision making units (or DMUs). Each DMU is evaluated on the basis of various input and output data (“less input and more output is better”) (see [61] for more information). In this example, the efficiency of six high schools is compared: Bloom (BL), Homewood (HW), New Trier (NT), Oak Park (OP), York (YK), and Elgin (EL). The input factors are “spending per pupil” and “not low income” (COST RICH), the output factors are “Writing score” and “Science score” (WRIT SCIN). Which schools are most efficient? The example is from [41].

Modeling Steps: Let the set of schools be $i, k \in I$ and the (input/output) factors $j \in J$ (the first two factors J^1 are input the others are output factors J^2 (with $J^1 \cap J^2 = J$). The input/output data are collected in the matrix $F_{i,j}$. The variables are the scores of each DMU ($SCORE_i$) and the weights $W_{i,j}$.

1. The sum of the scores is to be maximized:

$$\max \sum_i SCORE_i$$

2. The score of a DMU is the weighted output:

$$SCORE_i = \sum_{j \in J^2} F_{i,j} W_{i,j} \quad \text{forall } i \in I$$

3. The input of each DMU is normalized to 1 :

$$\sum_{j \in J^1} F_{i,j} W_{i,j} = 1 \quad \text{forall } i \in I$$

4. The output divided by the input on each DMU against each other must be smaller than 1:

$$\sum_{j \in J^2} F_{k,j} W_{i,j} \leq \sum_{j \in J^1} F_{k,j} W_{i,j} \quad \text{forall } i, k \in I$$

Solution: The scores of the schools are:

```
Score of BL : 1.000
Score of HW : 0.910
Score of NT : 0.962
Score of OP : 0.912
Score of YK : 1.000
Score of EL : 1.000
```

Hence, the most efficient schools are Bloom, Yorktown, and Elgin.

Further Comments: In LINGO, the syntax begins with MODEL and ends with END but they seemed to have no special functions besides to indicate the beginning or ending of the code. In LPL, by contrast, they are important because in this way one can modularize the code into multiple “sub-models”. LINGO’s syntax clear departs from the usual mathematical notation: Instead of SUM(J IN FACTOR) . . . it is @SUM(FACTOR(J) . . .) Equality and assignment operators are the same. Furthermore, it does not distinguish syntactically between active and passive indices (see my [Indexing Paper](#) for more information) – this is also the case, by the way, for GAMS. I find it essential to distinguish them syntactically, it sharpens the preciseness when creating and building models.

LPL code (run **deamod**)

```
model deamod "Data Envelopement Analysis";
set DMU,i,k := [BL HW NT OP YK EL] "Six schools";
FACTOR,j := [COST RICH WRIT SCIN] "input/output
factors";
parameter
// Inputs are "spending/pupil" and "not low income"
// Outputs are "Writing score" and "Science score"
NINPUTS := 2 "The first NINPUTS factors are inputs";
F{i,j} := [89.39 64.3      25.2    223
           86.25 99       28.2    287
           108.13 99.6     29.4    317
           106.38 96       26.4    291
           62.40 96.2     27.2    295
           47.19 79.9     25.5    222];
WGTMIN := .0005 "Min weight applied to every factor";
variable W{i,j} [WGTMIN..9999999] "weight";
SCORE{i};
```

```

maximize SC: sum{i} SCORE[i];
constraint
  A{i}: SCORE[i] = sum{j|j>NINPUTS} F[i,j]*W[i,j];
  B{i}: sum{j|j<=NINPUTS} F[i,j]*W[i,j] = 1;
  C{i,k}: sum{j|j>NINPUTS} F[k,j]*W[i,j] <=
            sum{j|j<=NINPUTS} F[k,j]*W[i,j];
  Write{i}('Score of %s : %5.3f\n', i, SCORE);
  Write('    %7s\n', {j} j);
  Write{i}('%3s %7.4f\n', i, {j} W);
end

```

LINGO code ([download deamod.lng](#))

```

MODEL:
! Data Envelopement Analysis (DEAMOD.LNG);
SETS:
  DMU/BL HW NT OP YK EL/: ! Six schools;
              SCORE; ! Each decision making unit has a;
              ! score to be computed;
  FACTOR/COST RICH WRIT SCIN/;
! There is a set of factors , input & output;
DXF(DMU, FACTOR):F, !F(I, J) = Jth factor of DMU I;
                    W; !Weights used computing DMU I's score;
ENDSETS
DATA:
  NINPUTS = 2; ! The first NINPUTS are inputs;
!      The inputs,      the outputs;
  F   =  89.39  64.3   25.2   223
        86.25  99     28.2   287
        108.13 99.6   29.4   317
        106.38  96     26.4   291
        62.40   96.2   27.2   295
        47.19   79.9   25.5   222;
  WGTMIN = .0005; ! Min weight applied to every factor
ENDDATA
!-----
!The Model. Everyone's score as high as possible;
MAX = @SUM( DMU: SCORE);
@FOR( DMU( I):
  SCORE( I) =
  @SUM(FACTOR(J)|J #GT# NINPUTS: F(I, J)* W(I, J));
  @SUM(FACTOR(J)|J #LE# NINPUTS: F(I, J)* W(I, J))=1;
@FOR( DMU( K):
  @SUM(FACTOR(J)|J #GT# NINPUTS: F(K, J) * W(I, J))<=
  @SUM(FACTOR(J)|J #LE# NINPUTS: F(K, J) * W(I, J));
  );
@FOR(DXF(I, J): @BND(WGTMIN, W(I, J), 9999999););
END

```

5.2.4 HEXALY (former LocalSolver)

hexaly is a relatively new kind of global optimization commercial solver. It provides APIs for Python, Java, C#, and C++ to model and solve your problems at hand. It also includes an powerful own proprietary modeling language. The optimizer combines state-of-the-art algorithms under the hood, it is very efficient for all kind of vehicle routing, scheduling, and other sequence problems and complements the best commercial MIP solvers, like Gurobi or Cplex. Large vehicle routing problems can be solved near optimally (see [CVRP](#)). It also can solve linear, integer, and non-linear problems without using derivatives, and contains lower bound algorithms (for minimizing problems) to get the optimality gap in the same way as MIP solver do. Models with logical constraints fits naturally into its syntax.

Its specially dedicated strength lies in its treatment of “permutation problems” (see [my Permutation Paper](#) for more information) which are ideally modeled and solved using hexaly. No other modeling system to my knowledge has this feature and capability. I use the classical cvrp (Capacitated Vehicle Routing Problem)

5.2.4.1 Capacitated Vehicle Routing Problem ([cvrp2](#))

— [Run LPL Code](#) , [HTML Document](#) —

Problem: A fixed fleet of delivery vehicles of uniform capacity must service known customer demands for a single commodity from a common depot at minimum transit cost. An concrete application is given in [cvrp](#). The models [cvrp-1](#), [cvrp-2](#), and [cvrp-3](#) give MIP-formulations of the problem. Formulate this problem as a permutation problem¹.

Modeling Steps: Let $i, j \in I = \{1, \dots, n - 1\}$ be a set of customer locations (n is the number of locations including the warehouse (or the depot) and let $k \in K = \{1, \dots, m\}$ be a set of trucks. Let the capacity of each truck be C_A , and let the demand quantity to deliver to a customer i be d_{m_i} . Furthermore, the distance between two customer i and j is $d_{i,j}$. Finally, the distance from the warehouse – the depot from where the trucks start – to each customer i is d_{w_i} .

The customers are enumerated with integers from 1 to $n - 1$. If there were one single truck (that must visits all customers), every permutation sequence of the numbers 1 to $n - 1$ would define a legal tour. Since we have m trucks, a sequence of a subset of 1 to $n - 1$ has to be assigned to each truck. Hence, each truck starts at the depot, visiting a (disjoint) subset of customers in a given order and returns to the depot.

¹ For a definition of *permutation problems* see [73].

The variables can now be formulated as a “partitioned permutation”. Example: if we had 3 trucks and 10 customers then the 3 subset sequences:

$$\{\{2, 3, 4\}, \{1, 9, 8, 5\}, \{6, 7, 10\}\}$$

defines a partitioning between the 3 trucks. It means, for example, that truck 1 starts at the depot visiting customers 2, 3, 4 in this order and returns to the depot. In LPL, we can declare these partitioned permutation with a permutation variable $x_{k,i} \in [1 \dots n - 1]$. For example: $x_{1,1} = 2$ means that the customer number 2 is visited by the truck 1 right after the depot, $x_{1,2} = 3$ means that the next customer (after 2) is 3, etc.

The unique constraint is the load of the truck that these subsets must fullfill (each subset sequence must be chosen in such a way that the capacity of the truck is larger than the cumulated demand of the customers that it visits):

$$\sum_{i|x_{k,i}} \text{dem}_{x_{k,i}} \leq CA \quad \text{forall } k \in K$$

We want to minimize the total travel distance of the trucks. Let cc_k be the size of the subset k (the numbers of customers that the truck k visits). Of course that number is variable and cannot be given in advance! Let routeDistances_k be the (unknown) travel distance of truck k , then we want to minimize the total distances (and in a first round the number of trucks):

$$\min \sum_k \text{routeDist}_k$$

where²

$$\text{routeDist}_k = \sum_{i \in 2..cc_k} d_{x_{k,i-1}, x_{k,i}} + \text{if}(cc_k > 0, dw_{x_{k,1}} + dw_{x_{k,cc_k}}) \quad \text{forall } k \in K$$

The term $dw_{x_{k,1}}$ denotes the distance of the truck k from the depot to the first customer, and $dw_{x_{k,cc_k}}$ is the distance of the truck tour k from the last customer to the depot, $d_{x_{k,i-1}, x_{k,i}}$ is the distance from a customer to the next – namely from customer $x_{k,i-1}$ to customer $x_{k,i}$ – and $\sum_{i \in 2..cc_k} \dots$ sums that distances of a tour k .

Further Comments: One of the most distiguished feature of LPL and the LSP (Hexaly language) is the use of variables within passive indices. This allows one to formulate an entire and practical important class of problems (the permutation problems, see [my Permutation Paper](#)). No other modeling system, to my knowledge, has this feature.

² The expression `if(boolExpr, Expr)` returns `Expr` if `boolExpr` is true else it returns 0 (zero).

A model in LSP is partitioned into, basically 4 functions: *input()*, where the data input is defined, *model()*, to declare the modeling structure, *param()* to specify the solver parameters, and *output()*, a procedure that describes the output.

Apart of *real*, *integer*, and *boolean* variables, LSP contains the *list* and *set* variables to specify permutations (see my paper above). In LPL, this kind of variables is declared as 1- and 2-dimensional integer variables with the keyword *alldiff*. Note also the graphical output of LPL and the concise code for reading the data.

LPL code (run **cvrp2**)

```
model cvrp "Capacitated Vehicule Routing Problem";
set i,j "customers";
      k "trucks";
parameter
  d{i,j} "distances";
  dw{i} "distance from/to warehouse";
  dem{i} "demand";
  CA "truck capacity";
alldiff x{k,i} 'partition' "customerSequences";
expression
  cc{k}: count{i} x;
  trucksUsed{k}: cc[k] > 0;
  routeDistances{k}:
    sum{i in 2..cc} d[x[k,i-1],x[k,i]]
    + if(cc[k]>0, dw[x[k,1]] + dw[x[k,cc[k]]]);
  nbTrucksUsed: sum{k} trucksUsed[k];
  totalDistance: sum{k} routeDistances[k];
constraint CAP{k}: sum{i|x} dem[x[k,i]] <= CA;
minimize obj1: nbTrucksUsed;
minimize obj2: totalDistance;
end
```

Hexaly code (download **cvrp.lsp**)

```
use io;
function input() {
  readInputCvRP();
  if (nbTrucks == nil) nbTrucks = getNbTrucks();
  computeDistanceMatrix();
}

function model() {
  customersSequences[k in 1..nbTrucks] <- list(
    nbCustomers);
  constraint partition[k in 1..nbTrucks](
    customersSequences[k]);
  for [k in 1..nbTrucks] {
    local sequence <- customersSequences[k];
```

```

local c <- count(sequence);
trucksUsed[k] <- c > 0;
routeQuantity <- sum(0..c-1, i => demands[sequence[i]]);
];
constraint routeQuantity <= truckCapacity;
routeDistances[k] <- sum(1..c-1, i => distanceMatrix[
    sequence[i - 1]][sequence[i]])
+ (c > 0 ? (distanceWarehouse[sequence[0]] +
    distanceWarehouse[sequence[c - 1]]) : 0);
}
nbTrucksUsed <- sum[k in 1..nbTrucks](trucksUsed[k]);
totalDistance <- sum[k in 1..nbTrucks](routeDistances[k]);
];
minimize nbTrucksUsed;
minimize totalDistance;
}

```

LPL input data model for cvrp2

```

model data;
set h; //h is i plus 1, 1 is depot (warehouse)
parameter de{h}; n;m; X{h};Y{h}; string typ; dum;
Read('A-n32-k5.vrp,%1;-1:DIMENSION',dum,dum,n);
Read('%1;-1:EDGE_WEIGHT_TYPE',dum,dum,typ);
if typ<>'EUC_2D' then
    Write('Only EUC_2D is supported\n'); return 0;
end;
Read('%1;-1:CAPACITY',dum,dum,CA);
Read{h}('%1:NODE_COORD_SECTION:DEMAND_SECTION', dum,X,Y
    );
Read{h}('%1:DEMAND_SECTION:DEPOT_SECTION', dum,de);
m:=Ceil(sum{h} de/CA);
k:=1..m;
i:=1..n-1;
d{i,j}:=Round(Sqrt((X[i+1]-X[j+1])^2+(Y[i+1]-Y[j+1])^2)
    );
dem{i}:=de[i+1];
dw{i}:=Round(Sqrt((X[i+1]-X[1])^2+(Y[i+1]-Y[1])^2));
end

```

LPL output model for cvrp2

```

model output friend data;
parameter y{k,i};
{k} (y[k,1]:=1, y[k,cc+2]:=1,{i|i<=cc} (y[k,i+1]:=x+1))
    ;
Draw.Scale(5,5);
{k,i in 1..cc+1} Draw.Line(X[y],Y[y],X[y[k,i+1]],Y[y[k,
    i+1]],k+3,3);
{k,i in 1..cc+2} Draw.Circle(y&'',X[y],Y[y],2,1,0);
end

```

Hexaly model input code

```

function param() { .... }
function output() { .... }

function readInputCvrp() {
    local inFile = io.openRead(inFileName);
    local nbNodes = 0;
    while (true) {
        local str = inFile.readString();
        if(str.startsWith("DIMENSION")) {
            if (!str.endsWith(":")) str=inFile.readString();
            nbNodes = inFile.readInt();
            nbCustomers = nbNodes - 1;
        } else if ((str.startsWith("CAPACITY"))) {
            if (!str.endsWith(":")) str=inFile.readString();
            truckCapacity = inFile.readInt();
        } else if((str.startsWith("EDGE_WEIGHT_TYPE"))) {
            if (!str.endsWith(":")) str=inFile.readString();
            local weightType = inFile.readString();
            if (weightType != "EUC_2D") throw ("Edge_Weight_
                Type_" + weightType + "_is_not_supported_
                only_EUC_2D");
        } else if(str.startsWith("NODE_COORD_SECTION")) {
            break;
        } else { local dump = inFile.readLine(); }
    }
    //nodeX and nodeY are indexed by original data indices
    (1 for depot)
    for[n in 1..nbNodes] {
        if (n != inFile.readInt()) throw "Unexpected_index";
    }
}

```

The output graph of LPL

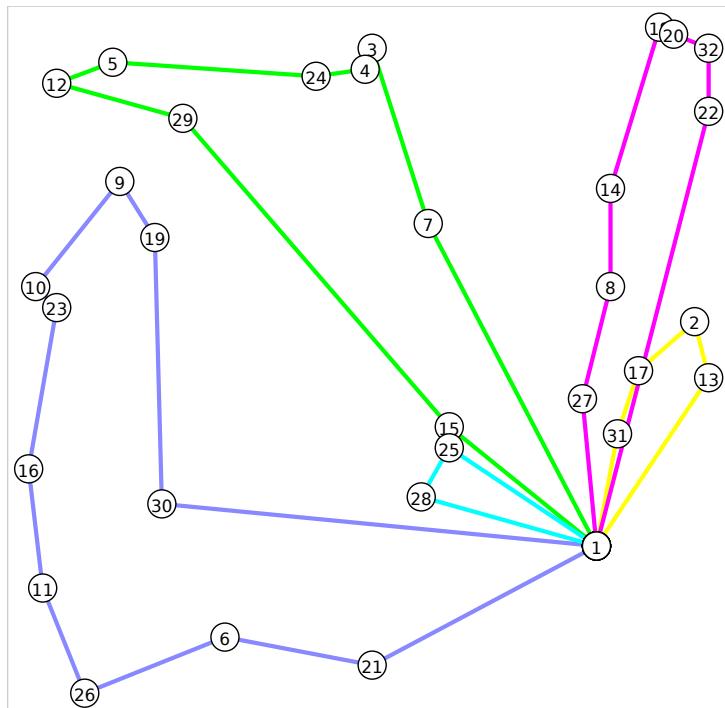


Figure 5.2: Optimal Solution of “A-n32-k5.vrp”

Hexaly input code (continued)

```

nodesX[n] = round(inFile.readDouble());
nodesY[n] = round(inFile.readDouble());
}
dump = inFile.readLine();
if (!dump.startsWith("DEMAND_SECTION")) throw "Expected
    keyword_DEMAND_SECTION";
for[n in 1..nbNodes] {
    if(n!=inFile.readInt()) throw "Unexpected_index";
    local demand = inFile.readInt();
    if (n == 1) {
        if (demand != 0) throw "expected_demand_for_depot_
            is_0";
    } else {
        demands[n-2] = demand; // demands is indexed by
            customers
    }
}
dump = inFile.readLine();
if (!dump.startsWith("DEPOT_SECTION")) throw "Expected_
    keyword_DEPOT_SECTION";

```

```

local warehouseId = inFile.readInt();
if (warehouseId != 1) throw "Warehouse_id_is_supposed_
    to_be_1";
local endOfDepotSection = inFile.readInt();
if (endOfDepotSection != -1) throw "Expecting_only_one_
    warehouse,_more_than_one_found";
}

function computeDistanceMatrix() { .... }
function getNbTrucks() { .... }

```

5.2.5 MiniZinc

MiniZinc is an open-source, solver-independent constraint modeling language. The [MiniZinc Software](#) includes also the solvers Gecode, Chuffed, COIN-OR CBC, and a Gurobi interface (see [45]). One can also model linear or non-linear optimization problems. Like other algebraic modeling languages, MiniZinc contains sets (defined as arrays), parameter, variables, constraints and solve instructions to formulate a model in a purely declarative way. As a language for constraint programming, constraints can contain Boolean operators, such as *and* (\wedge), *or* (\vee), *imply* (\rightarrow), or *not equal* (\neq) operators, etc. One of the most interesting and prominent feature of MiniZinc (and of other constraint programming systems) is its library of *global constraints* (see for example [Global Constraint Catalog](#)). Without going into the details here, two examples are given: (1) The *alldifferent* function constrains a set of (integer) variables to be all different from each other. (2) The *at_most* function requires at most n variables in a set of variables to take the given value v . Some of these global constraints could be expressed in a common mathematical notation, but – depending on the solver used – this would be inefficient. Given a set of integer variables x_i with $i, j \in I$, the *alldifferent* constraint, for example, could be expressed as

$$x_i \neq x_j \quad \text{forall } i, j \in I, j > i$$

The *atmost* constraint³ could be expressed as:

$$\sum_{i \in I} (x_i = v) = n$$

Expressing these constraints using a function name has the advantage that the constraint programming system recognizes them and can apply special and more efficient algorithms. MiniZinc contains many global constraints (see the [Reference Manual of MiniZinc](#) and it is the real strength of this language).

Two models are given to show *Boolean modeling* and the use of *alldifferent* constraint in MiniZinc.

³Note that LPL's **atmost** keyword does not correspond to this global constraint

5.2.5.1 Sudoku Puzzle (`sudokuM`)

— Run LPL Code , HTML Document –

Problem: The Sudoku problem is explained in another model (see `sudoku`).

Modeling Steps: The implementation for this model is explained elsewhere (see `sudokuInt`).

Further Comments: MiniZinc contains and uses the global constraint *alldifferent* (LPL uses the *Alldiff* function) to specify the constraints on the integer variables. MiniZinc uses ranges and array to specify multi-dimensional data sets or variables.

LPL code (run `sudokuM`)

```
model Sudoku "Sudoku Puzzle";
parameter n:=3;
set i,j := 1..n^2;
set g,h:=1..n; u,v:=1..n;
integer parameter start{i,j} := [
  5, 3, 1, 0, 0, 9, 6, 2, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 6, 0, 9, 4,
  0, 9, 6, 0, 3, 8, 1, 0, 0,
  0, 0, 0, 0, 0, 0, 3, 0, 0,
  7, 0, 0, 6, 0, 1, 0, 4, 0,
  0, 6, 0, 8, 0, 0, 4, 0, 0,
  1, 0, 5, 0, 2, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0];
integer variable X{i,j} [1..n^2];
constraint
  A{i}: Alldiff({j} X[i,j]);
  B{j}: Alldiff({i} X[i,j]);
  C{g,h}: Alldiff({u,v} X[(g-1)*n+u, (h-1)*n+v]);
  D{i,j|start}: X[i,j] = start[i,j];
solve;
Write{i}('%3d\n', {j} X);
end
```

MiniZinc code (download `sudokuM.mzn`)

```
set of int: RANGE = 1..9;
array[RANGE, RANGE] of int: start;
array[RANGE, RANGE] of var RANGE: X;
array[RANGE] of RANGE: first_i = [((k-1) div 3)*3 + 1| k
  in RANGE]; % first i in each square
array[RANGE] of RANGE: first_j = [((k-1) mod 3)*3 + 1| k
  in RANGE]; % first j in each square
start = [| 5, 3, 1, 0, 0, 9, 6, 2, 0,
          | 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```

| 0, 0, 0, 0, 0, 6, 0, 9, 4,
| 0, 9, 6, 0, 3, 8, 1, 0, 0,
| 0, 0, 0, 0, 0, 0, 3, 0, 0,
| 7, 0, 0, 6, 0, 1, 0, 4, 0,
| 0, 6, 0, 8, 0, 0, 4, 0, 0,
| 1, 0, 5, 0, 2, 0, 0, 0, 0,
| 0, 0, 0, 0, 0, 0, 0, 0 |];

```

```

include "alldifferent.mzn";
constraint forall(i in RANGE) (alldifferent([X[i,j] | j in RANGE]));
constraint forall(j in RANGE) (alldifferent([X[i,j] | i in RANGE]));
constraint forall(k in RANGE) (alldifferent([X[i,j] | i in first_i[k]..first_i[k]+2, j in first_j[k]..first_j[k]+2]));
constraint forall(i,j in RANGE where start[i,j]!=0) (X[i,j] = start[i,j]);
output [show([X[i,j] | j in RANGE]) ++ "\n" | i in RANGE];

```

```

solve satisfy;

```

5.2.5.2 Light Up Puzzle (**lightup1**)

— Run LPL Code , HTML Document –

Problem: The Light-Up puzzle consists of a rectangular grid of squares which are white or black. Every black filled square contains a number from 0 to 4, or may have no number. The aim is to place lights in the white squares so that

1. Each white square is illuminated, that is, it can see a light through an uninterrupted horizontal or vertical line of white squares.
2. No two lights can see each other.
3. The number of lights adjacent to a numbered filled square is exactly the number in the filled square.

The puzzle is from [MiniZinc Docs](#). An example of a puzzle is given in Figure 5.3.

Modeling Steps: For the size $w \times h$ (width/height) of the grid, we define a set $j, j \in W = \{1, \dots, w\}$ and $i, i \in H = \{1, \dots, h\}$. A parameter $b_{i,j} \in [-1, \dots, 5]$ is declared for every square (i, j) in the grid: $b_{i,j} = -1$ means the square is white, $b_{i,j} = 5$ means the black square is empty, otherwise ($b_{i,j} \in [0, \dots, 4]$) the square is black and contains the digit $b_{i,j}$.

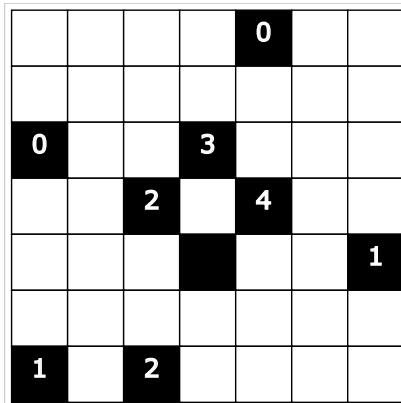


Figure 5.3: A Light Up Puzzle

Furthermore, a 4-dimensional relationship, called visible, $V_{i,j,i1,j1}$ is defined which is *true* if square $(i1, j1)$ is visible from square (i, j) , otherwise $V_{i,j,i1,j1}$ is *false*.

We want to know in which squares to put a light: a binary variable $x_{i,j}$ is introduced, it is = 1, if a light has to be placed in square (i, j) , otherwise it is = 0.

There are four constraints:⁴

(1) A black square does not have a light:

$$x_{i,j} = 0 \quad \text{forall } i \in H, j \in W, b_{i,j} \neq -1$$

(2) For a black numbered square (i, j) , the number of lights in the four neighbours (left/right, top/down) must correspond to the number in the square:

$$\sum_{i1,j1|R(i,j,i1,j1)} x_{i1,j1} = b_{i,j} \quad \text{forall } i \in H, j \in W, b_{i,j} \in [0, \dots, 4]$$

$$\text{where } R(i,j,i1,j1) = i-1 \leq i1 \leq i+1 \wedge j-1 \leq j1 \leq j+1 \wedge |i1-i| + |j1-j| = 1$$

(3) For all white squares (i, j) that are visible from $(i, j1)$ or from $(i1, j)$, either $(i, j1)$ or $(i1, j)$ (or both) must have a light:

$$\bigvee_{j1|V_{i,j,i,j1}} x_{i,j1} \vee \bigvee_{i1|V_{i,j,i1,j}} x_{i1,j} \quad \text{forall } i \in H, j \in W, b_{i,j} = -1$$

(4) For two different squares (i, j) and $(i1, j1)$ that are visible from each other, only one can have a light:

$$x_{i,j} \text{ nand } x_{i1,j1} \quad \text{forall } (i,j, i1, j1) \in V, i \neq i1, j \neq j1, b_{i,j} = b_{i1,j1} = -1$$

⁴Note: “ x nand y ” and “ x nor y ” mean $\overline{x \wedge y}$ and $\overline{x \vee y}$

Solution: The data file for LPL can be downloaded from [lightup1.txt](#). The solution for this data is given in Figure 5.4, generated by the output model of LPL (see below).

Further Comments: LPL and MiniZinc are similar in the notation of a logical constraint (both use exist (exists), and (\wedge), or (\vee) to express the constraint. In contrast to MiniZinc and depending on the the solver used, LPL translates the logical constraints into purely mathematical statements if using a MIP solver.

LPL code (run [lightup](#))

```

model lightup1 "Light Up Puzzle";
  set i,i1,H; // board height
  set j,j1,W; // board width
  parameter b{i,j} [-1..5]; // board
    E := -1; // empty square
    F := 5; // empty black square
  set visible{i,j,i1,j1};
  binary variable x{i,j}; // is there a light
  constraint C1{i,j|b<>E}: x=0;
  constraint C2{i,j|b=E nor b=F}:
    sum{i1,j1|i-1<=i1<=i+1 and j-1<=j1<=j+1 and
      Abs(i1-i)+Abs(j1-j)=1} x[i1,j1] = b[i,j];
  constraint C3{i,j|b=E}:
    exist{j1|visible[i,j,i,j1]} x[i,j1] or
    exist{i1|visible[i,j,i1,j]} x[i1,j];
  constraint C4{visible[i,j,i1,j1]|i<>i1 or j<>j1}:
    x[i,j] nand x[i1,j1];
  solve;
end

model data "the data model";
  parameter h; w;
  Read('lightup1.txt',h,w);
  i:=1..h; j:=1..w;
  Read{i}(' ,';1',{j} b);
  visible{i,j,i1,j1} :=
    (i=i1) and and{j2 in W|Min(j,j1)<=j2<=Max(j,j1)} (b[i,j2]
      ]=E) or
    (j=j1) and and{i2 in H|Min(i,i1)<=i2<=Max(i,i1)} (b[i2,j
      ]=E);
end

```

MiniZinc code ([download lightup1.mzn](#))

```

int: h; set of int: H = 1..h; % board height
int: w; set of int: W = 1..w; % board width
array[H,W] of -1..5: b; % board
int: E = -1; % empty square

```

```

set of int: N = 0..4; % filled and numbered square
int: F = 5; % filled unnumbered square

test visible(int: i1, int: j1, int: i2, int: j2) =
  ((i1 == i2) /\ forall(j in min(j1,j2)..max(j1,j2))
   (b[i1,j] == E)) \/
  ((j1 == j2) /\ forall(i in min(i1,i2)..max(i1,i2))
   (b[i,j1] == E));
array[H,W] of var bool: x; % is there a light
constraint forall(i in H, j in W where b[i,j] != E) (x[i,j]
  ] == false);
constraint forall(i in H, j in W where b[i,j] in N) (
  bool_sum_eq([ x[i1,j1] | i1 in i-1..i+1,
    j1 in j-1..j+1 where abs(i1-i) + abs(j1-j) == 1
    /\ i1 in H /\ j1 in W ], b[i,j]));
constraint forall(i in H, j in W where b[i,j] == E) (
  exists(j1 in W where visible(i,j,i,j1)) (x[i,j1]) \/
  exists(i1 in H where visible(i,j,i1,j)) (x[i1,j]));
constraint forall(i1,i2 in H, j1,j2 in W where
  (i1 != i2 /\ j1 != j2) /\ b[i1,j1] == E
  /\ b[i2,j2] == E /\ visible(i1,j1,i2,j2))
  (not x[i1,j1] \/ not x[i2,j2]);
solve satisfy;

```

LPL output model is as follows

```

model output;
  Write{i}('%2s \n', {j} if(b<>E,b&' ',x=1,'L','.'));
  Draw.Scale(40,40);
  Draw.DefFont('verdana',18,1,1,1);
  --{i,j} Draw.Rect(if(b=E or b=F,'','&b),j,i,1,1,if(b
    ==-1,1,0),0);
  {i,j} Draw.Rect(if(b=E or b=F,'','&b),j,i,1,1,if(b
    ==-1,6,0),0);
  {i,j|x} Draw.Circle(j+.5,i+.5,0.4,1,0);
end

```

The output model generates the Figure 5.4

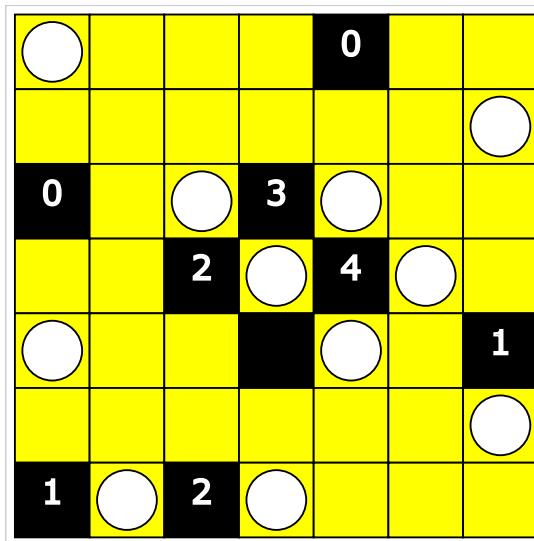


Figure 5.4: Solution of the Lightup Puzzle

```

output [ if b[i,j] != E then show(b[i,j])
  else if fix(x[i,j]) then "L" else "." endif
endif ++ if j == w then "\n" else "_" endif
| i in H, j in W];
  
```

MiniZinc data file (download [lightup1.dzn](#))

```

h = 7;
w = 7;
b = [| -1,-1,-1,-1, 0,-1,-1
      | -1,-1,-1,-1,-1,-1,-1
      | 0,-1,-1, 3,-1,-1,-1
      | -1,-1, 2,-1, 4,-1,-1
      | -1,-1,-1, 5,-1,-1, 1
      | -1,-1,-1,-1,-1,-1,-1
      | 1,-1, 2,-1,-1,-1,-1 |];
  
```

5.2.6 AIMMS

AIMMS is a modeling software that features a mixture of declarative and imperative programming styles. Optimization models can be formulated in the style of algebraic modeling languages, with sets and indices, as well as multidimensional parameters, variables and constraints. Units of measurement are natively supported in the language. AIMMS has a interface to many linear and non-linear solvers and is designed for large-scale optimization and

scheduling-type problems. The syntax is very verbose and each element consists of a number of attributes, it is like an entry in a database table with the attributes as fields.

5.2.6.1 A Non-transitive Relation (**dice**)

— Run LPL Code , HTML Document –

Problem: In the dice problem a set of three dice has to be designed by assigning an integer number to each face such that on average dice 1 beats dice 2, dice 2 beats dice 3 and dice 3 beats dice 1. The goal is to maximize the number of total wins on average. The dice problem has many solutions. The problem is from Robert A Bosch, Monochromatic Squares, Optima, MP Society Newsletter, Vol 71, March 2004, page 6-7. A related problem is also implemented in **monochrom**.

Modeling Steps: Let $f, fp \in \text{Faces} = \{1, \dots, 6\}$ be the faces on a dices and let $d \in \text{Dices} = \{1, \dots, 3\}$ be the set of dices. The lowest face value is $f_{lo} = 1$, and the highest is $f_{up} = |\text{Dice}| \cdot |\text{Faces}|$. The variable Obj is the number of wins.

$y_{i,f,fp}$ is defined as follows: if face f of dice i wins (value 1) or loses (value 0) against face fp of dice $i + 1$ (circular).

1. Count the wins of all dices:

$$\sum_{f, fp} y_{i,f,fp} = \text{Obj}$$

2. The second constraint is the definition of $y_{i,f,fp}$, that is, face f of dice i wins (value 1) or loses (value 0) against face fp of dice $i + 1$:

$$x_{i,f} + (f_{up} - f_{lo})(1 - y_{i,f,fp}) \geq x_{i,(i+1),fp} + 1 \\ \text{forall } i \in \text{Dices}, f, fp \in \text{Faces}$$

3. All faces on a dice must have different numbers:

$$x[i, f-1] + 1 \leq x[i, f] \quad \text{forall } i \in \text{Dices}, f \in \text{Faces} - \{\text{face1}\}$$

4. We want to maximize the number of wins:

$$\max \sum_{f, fp} y_{i,f,fp}$$

LPL code (run **dice**)

```

model dice "A non-transitive relation";
set
  Faces,f,fp:=[1..6] "Faces on a dice";
  Dices,i := [1..3] "Number of dices";
parameter
  flo := 1      "Lowest face value";
  fup := #i*f "Highest face value";

variable
  Obj           "Number of wins";
  x{i,f} [flo..fup] "Face value of dice";
  binary y{i,f,fp} "=1, if f beats fp (MatchOutcome)";

constraint
  Wins{i}: sum{f,fp} y[i,f,fp] = Obj "Count the wins";
  Fval{i,f,fp}: x[i,f] + (fup-flo)*(1-y[i,f,fp])
    >= x[if(i<#i,i+1,1),fp] + 1      "non-trans.
    relation";
  Fa{i,f|f>1}: x[i,f-1]+1 <= x[i,f] "diff face values";
  Fix: x['dice1','face1'] = flo      "fix one value";
maximize N: Obj;
end

```

AIMMS code ([download dice.ams](#))

```

Model Dice_Model {
  Comment: {
    "Finding_multiple_solutions..."
  }
  DeclarationSection Declaration_Model {
    Set Faces {
      Text: "Faces_on_a_dice";
      Index: f, fp;
      InitialData: data { face1 .. face6 };
    }
    Set Dice {
      Index: i;
      InitialData: data { dice1 .. dice3 };
    }
    Parameter LowestFaceValue {
      Text: "Lowest_face_value";
      InitialData: 1;
    }
    Parameter HighestFaceValue {
      Text: "Highest_face_value";
    }
    Variable Obj {
      Comment: "Number_of_wins.";
    }
    Variable FaceValue {
  }

```

```

    IndexDomain: (i,f);
    Range: binary;
    Comment: "Face_value_of_dice.";
}
Variable MatchOutcome {
    IndexDomain: (i,f,fp);
    Range: binary;
    Comment: {
        "Binary_variable_indicating_..."
    }
}

```

... continuing AIMMS code ...

```

Constraint WinsConstraint {
    IndexDomain: i;
    Definition: Obj = sum( (f,fp), MatchOutcome(i
        ,f,fp) );
    Comment: "Count_the_wins_of_all_dice.";
}
Constraint MatchOutcomeConstraint {
    IndexDomain: (i,f,fp);
    Definition: FaceValue(i,f) + (
        HighestFaceValue-LowestFaceValue)*(1-
        MatchOutcome(i,f,fp)) >= FaceValue(i+1,
        fp) + 1;
    Comment: {
        "Constraint_that_determines_..."
    }
}
Constraint FaceValuesConstraint {
    IndexDomain: (i,f) | (f-1) in faces;
    Definition: FaceValue(i,f-1) + 1 <= FaceValue
        (i,f);
    Comment: "Constraint_to_enforce_...";
}
MathematicalProgram DiceMP {
    Objective: Obj;
    Direction: maximize;
    Constraints: AllConstraints;
    Variables: AllVariables;
    Type: MIP;
}
}
Procedure MainInitialization {
    Body: {
        HighestFaceValue := card(Dice) * card(faces);

        FaceValue.lower(i,f) := LowestFaceValue;
    }
}
```

```

        FaceValue.upper(i,f) := HighestFaceValue;

        FaceValue.lower('dice1','face1') :=
            LowestFaceValue;
        FaceValue.upper('dice1','face1') :=
            LowestFaceValue;
        FaceValue.level('dice1','face1') :=
            LowestFaceValue;

        SolutionMethod := 'normal';
    }
}
Procedure MainExecution {
    Body: {
        empty Objs, FaceValues;

        ! Optimal objective value: 21.
        solve DiceMP;

        Objs(1) := Obj;
        FaceValues(i,f,1) := FaceValue(i,f);
    }
    Comment: {
        "Normal_solve;_returns_one_solution."
    }
}
Procedure MainTermination {
    Body: {
        return 1;
    }
}
}

```

5.2.7 MOSEL

Mosel Language can be thought of as both a modeling language and a programming language. Like other modeling languages it offers the required facilities to declare and manipulate problems, decision variables, constraints and various data types and structures like sets and arrays. On the other hand, it also provides a complete set of functionalities proper to programming languages: it is compiled and optimized, all usual control flow constructs are supported and can be extended by means of modules. Although a general modeling system, it is closely interfaced with its own XPress optimizer. A link to a separate constraint programming solver **Kalis** is offered too. Many modeling examples in Mosel are available from the [Fico Site](#).

I use a small production planning model to illustrate the syntax of Mosel.

5.2.7.1 A Tiny Planning Model ([tiny-initial](#))

— Run LPL Code , HTML Document —

Problem: Pochet/Wolsey [56] introduce the production planning models by a tiny example: A manufacturer plans to produce a high-tech bicycle. In each month at most one batch is produced in order to satisfy the demand. Sales forecast for the product is given. What should the batch sizes be in each month in order to satisfy the demand and to minimize costs? More information is given in model [tiny](#).

Modeling Steps: We introduce a set $t \in T = \{1, \dots, 8\}$ to specify the planning horizon. Let $q = 5000$ be the set-up cost of a batch, $p = 100$ the production cost of a unit, $s_0 = 200$ the initial stock, and $h = 5$ the storage cost of a unit in stock per month. The demand in each months is given as d_t . We want to know the quantities $x_t \geq 0$ of bicycles produced in each month. Furthermore, we want to know the quantity in stock at the end of each month ($s_t \geq 0$). The quantity in stock at the end should be zero, hence $s_8 = 0$. Finally, a binary variable y_t is needed to specify whether a batch must be opened ($= 1$) in month t or not ($= 0$). The model then is as follows:

$$\begin{aligned} \min \quad & \sum_t px_t + \sum_t hs_t + \sum_t qy_t \\ \text{s.t.} \quad & s_{t-1} + x_t = d_t + s_t \quad \text{forall } t \in T \\ & x_t \leq My_t \quad \text{forall } t \in T \\ & s_0 = s_0, s_{|T|} = 0 \\ & x_t \geq 0, s_t \geq 0, y_t \in \{0, 1\} \quad \text{Sucht } t \in T \end{aligned}$$

Costs are to be minimized. They are composed by production costs the storage costs, and the set-up costs. The first constraint defines the balance from period to period: stock at the beginning of t plus production equal demand plus stock at the end of t . The second constraint defines the connection between production and batch opening: if no batch is opened ($y_t = 0$), then no bike is produced ($x_t = 0$).

LPL code (run [tiny-initial](#))

```
model tiny "A Tiny Planning Model";
set t,k      "Range of time";
parameter q "Setup cost";
p            "Price per unit";
s0           "Initial Inventory";
h            "Unit storage cost";
d{t}         "Demand";
variable x{t} "Production";
s{t}         "Storage";
binary variable y{t} "=1, batch opened in t";
```

```

constraint
    dem_sat{t}: if(t=1,s0,s[t-1]) + x[t] = d[t] + s[t];
    vub{t}: x[t] <= (sum{k} d[k])*y[t];
    endStock: s[#t]=0;
minimize cost:
    sum{t} (p*x[t]+q*y[t]+if(t<#t,h,h/2)*s[t]);

model data;
    q:=5000; p:=100; s0:=200; h:=5;
    d{t} := /Jan 400 Feb 400 Mar 800 Apr 800
           May 1200 Jun 1200 Jul 1200 Aug 1200/;
end

model output;
    Write('      %7s \nDemand %7d \nProduc %7d \
          \nInvent %7d \n\n P cost %7d \nI cost %7d \
          \nSet-up %7d \n\n Total costs: %7d',
          {t}t, {t}d, {t}x, {t}s, {t}p*x, {t}h*s, {t}q*y, cost
        );
end
end

```

MOSEL code (download [tiny-initial.mos](#))

```

model tiny                      ! Start a new model
uses "mmxprs", "mmsystem"       ! Load the optimizer library
options noimplicit

declarations
    EPS=1e-6                      ! Zero tolerance
    T=8                            ! Number of time periods
    RT=1..T                         ! Range of time

    DEMAND: array(RT) of integer ! Demand per period
    SETUPCOST: integer           ! Setup cost per period
    PRODCOST: integer            ! Production cost per period
    INV COST: integer            ! Production cost per period
    STOCKINI: integer            ! Production cost per period

    D: array(RT,RT) of integer ! Demand per period

    s: array(RT) of mpvar     ! Inventory in period t
    x: array(RT) of mpvar     ! Production in period t
    y: array(RT) of mpvar     ! Setup in period t

    balance: array(RT) of linctr
    production: array(RT) of linctr
    mincost : linctr
    vil: array(RT) of linctr

```

```
end-declarations
```

```
DEMAND := [400, 400, 800, 800, 1200, 1200, 1200, 1200]
```

...continuing MOSEL code ...

```
SETUPCOST:= 5000
PRODCOST:= 100
INV COST:= 5
STOCKINI:= 200

forall(r in RT, t in RT) do
    if (r <= t) then
        D(r,t) := sum(k in r..t) DEMAND(k)
    else
        D(r,t) := 0
    end-if
end-do

! Objective: minimize total cost
mincost:=
    sum(t in RT ) (if(t<T, INV COST, INV COST/2)) *s(t) +
    sum(t in RT) (SETUPCOST*y(t) + PRODCOST*x(t))

forall(t in RT) production(t) := x(t) <= D(t,T)*y(t)

forall(t in RT) balance(t):=
    if (t=1, STOCKINI,s(t-1))+x(t) = DEMAND(t) + s(t)

forall(t in RT) y(t) is_binary ! Vars setup are 0/1

setparam("XPRS_VERBOSE", 1)      !
setparam("XPRS_MIPLOG", 3)       !
setparam("XPRS_PRESOLVE", 1)     ! Disable PRESOLVE
setparam("XPRS_CUTSTRATEGY", 0)   ! Disable autom. cuts

minimize(mincost)
! minimize(XPRS_LIN, mincost)

writeln
forall(t in RT)
    writeln("Period_", t, ":_prod_", getsol(x(t)), "_(" demand:
        "_", DEMAND(t),
        ",_unit_cost:_", PRODCOST, "),_setup_", getsol(
            y(t)),
        "_(" cost:_", SETUPCOST, ",_stock:_", getsol(s(t))
        ), ")")

end-model
```

5.2.8 OPL

Optimization Programming Language (OPL) is an algebraic modeling language for mathematical optimization models, which makes the coding easier and shorter than with a general-purpose programming language. It is part of the CPLEX software package and therefore tailored for the IBM ILOG CPLEX and IBM ILOG CPLEX CP (constraint programming) optimizers. The separate IBM ILOG Script for OPL is an embedded JavaScript implementation that provides the “non-modeling” expressiveness of OPL.

To display some aspects of the OPL language, a multi-commodity transportation is used.

5.2.8.1 Multi-commodity Transportation ([transp3](#))

— [Run LPL Code](#) , [HTML Document](#) –

Problem: This is a simple multi-commodity transprtation problem: various products have to be transported from a supply location to different demand center. Unit transportation cost, supply quantity, transport capacity, and demand are given. How much of a product must be transported from a supply location to a demand location at minimial costs?

Modeling Steps: Given a set of locations $i, j \in I$ and a set of products $p \in P$. All given routings and the cost are given as $R_{p,i,j}$ and $c_{p,i,j}$, that is, if a product p is transported from i to j then $\text{route}_{p,i,j}$ is true otherwise false. Supply quantity from location i and demand quantity to j of a product p are given in $s_{p,i}$ and $d_{p,j}$. From the route the supply locations and the destination location can be derived: Orig_i and Dest_j , that is: if i is a supply location then Orig_i is true (otherwise false). In the same way all the feasible connections $\text{CO}_{i,j}$ can be derived from route. The variable is the quantity to be transported of a product p from i to j : $\text{Trans}_{p,i,j}$. The whole model is as follows:

$$\begin{aligned} \min & \sum_{(p,i,j) \in R} c_{p,i,j} \cdot \text{Trans}_{p,i,j} \\ \text{subject to} & \sum_{j \in R} \text{Trans}_{p,i,j} = s_{p,i} && \text{forall } p \in P, i \in \text{Orig} \\ & \sum_{i \in R} \text{Trans}_{p,i,j} = d_{p,j} && \text{forall } p \in P, j \in \text{Dest} \\ & \sum_{p \in R} \text{Trans}_{p,i,j} \leq \text{Capacity} && \text{forall } (i,j) \in \text{CO} \\ & \text{Trans}_{p,i,j} \geq 0 && \text{forall } (p,i,j) \in R \end{aligned}$$

Further Comments: OPL uses the ellipsis ... to specify the missing data in an otherwise structural model code. They are given in an separate data file. The

concept of *tuple* is used to relate data, like the concept of “record” or “struct” in programming languages. This is definitely a concept that is missing in the LPL language. Due to this feature it is also possible to create compound (sparse) sets. However, I find LPL’s compound sets syntax more straightforward and shorter.

LPL model code (run **transp3**)

```
model transp3 "Multi-commodity Transportation";
  set i,j,Cities;
  set p,Products;
  parameter Capacity;
  set connection{i,j};
  set route{p,i,j};
  parameter Supply{p,i};
  parameter Demand{p,j};
  parameter Cost{p,i,j};
  set Orig{i}; Dest{j};
  variable Trans{p,i,j|route};
  minimize obj:sum{route[p,i,j]} Cost*Trans;
  constraint ctSupply{p,Orig[i]}:
    sum{j in route} Trans = Supply;
  constraint ctDemand{p,Dest[j]}:
    sum{i in route} Trans = Demand;
  constraint ctCapacity{connection[i,j]}:
    sum{p in route} Trans <= Capacity;
end
```

LPL data model : (The data itself is in file **transp3.txt**)

```
model data;
  Read('transp3.txt,%1:Cities',{i} i);
  Read('%1:Products',{p} p);
  Read('%1:Capacity', Capacity);
  Read{p,i,j}('%1:Routes:Supply',p,i,j,route,Cost);
  Read{p,i}('%1:Supply:Demand',p,i,Supply);
  Read{p,j}('%1:Demand:Cost',p,j,Demand);
  connection{i,j}:=exist{p} route;
  Orig{i}:=exist{j} connection;
  Dest{j}:=exist{i} connection;
  {p} if(sum{Orig[i]} Supply <> sum{Dest[j]})  

    Demand , Write('Inconsistent\n'));
end;
```

OPL code (download **transp3.mod**)

```
{string} Cities =...;
{string} Products = ...;
float Capacity = ...;
tuple connection { string o; string d; }
tuple route {
```

```

string p;
connection e;
}
{route} Routes = ...;
{connection} Connections = { c | <p,c> in Routes };
tuple supply {
    string p;
    string o;
}
{supply} Supplies = { <p,c.o> | <p,c> in Routes };
float Supply[Supplies] = ...;
tuple customer {
    string p;
    string d;
}
{customer} Customers = {<p,c.d> | <p,c> in Routes};
float Demand[Customers] = ...;
float Cost[Routes] = ...;
{string} Orig[p in Products] = { c.o | <p,c> in Routes };
{string} Dest[p in Products] = { c.d | <p,c> in Routes };

{connection} CPs[p in Products] = { c | <p,c> in Routes
    };
assert forall(p in Products)
    sum(o in Orig[p]) Supply[<p,o>] == sum(d in Dest[p])
        Demand[<p,d>];

dvar float+ Trans[Routes];

constraint ctSupply[Products][Cities];
constraint ctDemand[Products][Cities];

minimize
    sum(l in Routes) Cost[l] * Trans[l];
subject to {
    forall( p in Products , o in Orig[p] )
        ctSupply[p][o]:
            sum( <o,d> in CPs[p] )
                Trans[< p,<o,d> >] == Supply[<p,o>];
    forall( p in Products , d in Dest[p] )
        ctDemand[p][d]:
            sum( <o,d> in CPs[p] )
                Trans[< p,<o,d> >] == Demand[<p,d>];
    forall(c in Connections)
        ctCapacity:
            sum( <p,c> in Routes )
                Trans[<p,c>] <= Capacity;
}
}

```

5.2.9 MATLAB

MATLAB is a programming platform designed specifically for engineers and scientists to analyze and design systems and products. The heart of MATLAB is the MATLAB language, a matrix-based multi-paradigm programming language and a numeric computing environment allowing the most natural expression of computational mathematics. MATLAB is one the most widespread mathematical system. Although MATLAB is intended primarily for numeric computing, a toolbox uses the MuPAD symbolic engine, another package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems. Another modeling approach to optimization with MATLAB is the [TOMLAB Optimization Environment](#). A free alternative of MATLAB is [Octave](#).

A small blending problem is given to show the optimization approach with MATLAB. Two implementations of the model are given to illustrate various aspects of the system.

5.2.9.1 Steel Blending ([mixing](#))

— [Run LPL Code](#) , [HTML Document](#) –

Problem: The problem is “to blend steels with various chemical compositions to obtain 25 tons of steel with a specific chemical composition. The result should have 5% carbon and 5% molybdenum by weight. The objective is to minimize the cost for blending the steel”. The problem is from [MatLab Doc](#).

Modeling Steps: A set $i \in I$ of different ingots of steel are available for purchase. Only one of each ingot type is available. Their weight is w_i , the carbon and molybdenum contents in percentage is given by r_i and m_i , and the cost per ton is c_i .

Furthermore, a set $j \in J$ of grades of alloy steel and one grade of scrap steel are available for purchase. Alloy and scrap steels can be purchased in fractional amounts. the carbon and molybdenum contents in percentage of each alloy is given by r_{aj} and m_{aj} , and the cost per ton is c_{aj} . The scrap content of the carbon and molybdenum in percentage is r_s and m_s and its cost per ton is c_s .

Let I_i be the number of ingots, A_j the fractional weight of each alloy, and S the weight of scrap. So, how many ingots and how much alloy and scrap must be used to produce W tons of steel with a $p\%$ of carbon and molybdenum while minimizing the total costs. The model can be formulated as follows:

$$\begin{aligned}
 \min \quad & \sum_i c_i I_i + \sum_j c a_j A_j + cs \cdot S \\
 \text{subject to} \quad & \sum_i w_i I_i + \sum_j A_j + S = W \\
 & \sum_i w_i r_i I_i + \sum_j r a_j A_j + rs \cdot S = pW \\
 & \sum_i m_i I_i + \sum_j m a_j A_j + ms \cdot S = pW \\
 & I_i \in [0, 1], A_j \geq 0, S \geq 0, \quad \text{forall } i \in I, j \in J
 \end{aligned}$$

Further Comments: In the first formulation, the MatLab formulation is copied one-to-one to LPL. MatLab is matrix-base. Indices are not needed. In LPL, passive indices can be omitted if the context allows it. MatLab does not make a distinction between parameters and expression, like LPL does. MatLab uses a package `optimproblem` to specify the problem.

The second formulation of both modeling languages is closer to their “style”. In MatLab, a single function `intlinprog` defines a complete integer linear model. The parameters are prepared in advance as data matrices. In LPL, the model consists of the common structure: set, parameter, variable, constraint and objective function.

LPL code (first formulation) (run `mixing`)

```

model mixing "Steel Blending";
set i,Ingot := [1..4];
set j, Alloy := [1..3];
variable alloys{j}; scrap;
binary ingots{i};
parameter weightIngots{i}:=[5,3,4,6];
uCostIngots{i}:=[350,330,310,280];
costIngots{i}:=weightIngots*uCostIngots;
costAlloys{j}:=[500,450,400];
costScrap:=100;
expression cost: sum{i} costIngots*ingots + sum{j}
    costAlloys*alloys + costScrap*scrap;
expression totalWeight: sum{i} weightIngots*ingots +
    sum{j} alloys + scrap;
parameter carbonIngots{i} := [5,4,5,3];
carbonAlloys{j} := [8,7,6];
carbonScrap := 3;
expression totalCarbon: sum{i} weightIngots*carbonIngots
    /100*ingots + sum{j} carbonAlloys/100*alloys +
    carbonScrap/100*scrap;
parameter molybIngots{i}:=[3,3,4,4];
molybAlloys{j} := [6,7,8];
molybScrap := 9;

```

```

expression totalMolyb: sum{i}weightIngots*molybIngots
    /100*ingots + sum{j}molybAlloys/100*alloys +
    molybScrap/100*scrap;
constraint conswt: totalWeight = 25;
    conscarb: totalCarbon = 1.25;
    consmolyb: totalMolyb = 1.25;
minimize C: cost;
    Writep(ingots,alloys,scrap,cost);
end

```

MatLab code (first formulation) (download [mixing.m](#))

```

prob = optimproblem;
ingots = optimvar('ingots',4,'Type','integer','LowerBound
    ',0,'UpperBound',1);
alloys = optimvar('alloys',3,'LowerBound',0);
scrap = optimvar('scrap','LowerBound',0);
weightIngots = [5,3,4,6];
costIngots = weightIngots.*[350,330,310,280];
costAlloys = [500,450,400];
costScrap = 100;
cost = costIngots*ingots + costAlloys*alloys + costScrap*
    scrap;

prob.Objective = cost;
totalWeight = weightIngots*ingots + sum(alloys) + scrap;
carbonIngots = [5,4,5,3]/100;
carbonAlloys = [8,7,6]/100;
carbonScrap = 3/100;
totalCarbon = (weightIngots.*carbonIngots)*ingots +
    carbonAlloys*alloys + carbonScrap*scrap;
molybIngots = [3,3,4,4]/100;
molybAlloys = [6,7,8]/100;
molybScrap = 9/100;
totalMolyb = (weightIngots.*molybIngots)*ingots +
    molybAlloys*alloys + molybScrap*scrap;

prob.Constraints.conswt = totalWeight == 25;
prob.Constraints.conscarb = totalCarbon == 1.25;
prob.Constraints.consmolyb = totalMolyb == 1.25;
[sol,fval] = solve(prob);

sol.ingots
sol.alloys
sol.scrap
fval

```

LPL code (second formulation) (run [mixing1](#))

```
| model mixing "Steel Blending I";
```

```

set i,Ingot := [1..4];
    j, Alloy := [1,2,3,scrap];
parameter weightIngots{i}:=[5,3,4,6];
    uCostIngots{i}:=[350,330,310,280];
    costAlloys{j}:=[500,450,400,100];
    carbonIngots{i} := [5,4,5,3];
    carbonAlloys{j} := [8,7,6,3];
    molybIngots{i}:=[3,3,4,4];
    molybAlloys{j} := [6,7,8,9];
variable A{j}; binary I{i};
constraint conswt:
    sum{i} weightIngots*I + sum{j} A = 25;
    conscarb: sum{i}weightIngots*carbonIngots/100*I + sum
        {j}carbonAlloys/100*A = 1.25;
    consmolyb: sum{i}weightIngots*molybIngots/100*I +
        sum{j}molybAlloys/100*A = 1.25;
minimize cost: sum{i} weightIngots*uCostIngots*I + sum{
    j} costAlloys*A;
    Writep(I,A,cost);
end

```

MatLab code (second formulation) (download [mixing1.m](#))

```

f = [350*5,330*3,310*4,280*6,500,450,400,100];
intcon = 1:4;

A = [];
b = [];

Aeq = [5,3,4,6,1,1,1,1;
    5*0.05,3*0.04,4*0.05,6*0.03,0.08,0.07,0.06,0.03;
    5*0.03,3*0.03,4*0.04,6*0.04,0.06,0.07,0.08,0.09];
beq = [25;1.25;1.25];

lb = zeros(8,1);
ub = ones(8,1);
ub(5:end) = Inf; % No upper bound on noninteger variables

[x,fval] = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub);

x,fval      % print the solution

```

5.2.10 Other Modeling Language Tools

I hope that I presented briefly the most important (optimization) modeling systems. Many other systems and tools exist.

- The modeling language **Zimpl** together with SCIP is bundled in the SCIP Optimization Suite. Academic licence is free.
- **MPL** is another algebraic modeling language that exists since a long-time.
- The **APMonitor** language is explained below together with Python package interface Gekko.
- The **Wolfram Language of Mathematica** integrates a full range of state-of-the-art local and global optimization techniques. The integrated programming language has a functional flavor.
- **Maple** is another all-in-one mathematical software. Its language is closer to a procedural language similar to C. It also contains a global optimization toolbox. Both Maple and Mathematica have an interactive documentation tool to write “executable” formatted mathematical documents.

5.3. Programming Languages

Python seems to be a very popular programming language for implementing optimization models at the present time. There exists many packages that facilitate the formulation of problems and the interface to solvers. Python is free and can be implemented from [download python](#). Another Python distribution with many packages already installed is [Anaconda](#). It is especially interesting when using Jupyter Notebook (see below). With the packages NumPy, Panda, SciPy, etc. Python is an ideal tool for scientific computation and data management. To solve models with the various packages – explained there after, the user need also install several solvers using the pip command (or the conda command for anaconda). A first impression comparing three Python packages is given in the article [Compares Python interface Pulp-Gurobi-Cplex](#).

For the purpose of this paper the free solver *glpk* (linear models), *ipopt* (non-linear models) and the commercial solver *cplex* and *gurobi* are used. Note that a licence is needed for the commercial solvers and separate installation of the solvers is needed too. To install the packages, which installs an interface to the solvers, one need the type on the command line (or the Python command line) :

```
pip install glpk
pip install ipopt
pip install cplex
pip install gurobipy
```

5.3.1 Gurobipy (Python – commercial)

Gurobipy is the modeling tool distributed by *Gurobi* together with and uniquely for the *Gurobi* solver. Documentation can be found at [Gurobi Python Quick-start](#), [gurobipy API](#), and [Python Models](#).

The package *gurobipy* specifies a model using the Python object structure. A model is initiated by the function *Model()*. Using the functions *addVars*, *addConstrs*, and *setObjective* a model can be specified. The concept of sets and multi-dimensional data are provided by the Python's own data structures of lists, tuples, and dictionaries. We use 3 models to illustrate the syntax: a small linear *netflow* model to show the basic syntax, a *tic-tac-toe* model to show integer models and indicator variables, and a *regression* model to display the recent vectorized notation.

5.3.1.1 Multi-commodity Flow (*netflow*)

— Run LPL Code , HTML Document –

Problem: Two products ('Pencils' and 'Pens') are to be transported from the factories in ('Detroit' and 'Denver') to the warehouses in ('Boston', 'New York', and 'Seattle') to satisfy the demand ('inflow[h,i]'). The flows on the transportation network must respect arc capacity constraints ('capacity[i,j]'). The objective is to minimize the sum of the arc transportation costs ('cost[i,j]'). The Python version is from the *Gurobi* examples.

Modeling Steps: The mathematical model is as follows: Let $h \in H$ be the set of commodities, $i, j \in I$ the set of locations (factories and warehouses). Let $(i, j) \in A$ be the arcs in the networks. The costs are $c_{i,j}$, the demand is $d_{h,i,j}$, and the arc capacity is $C_{i,j}$. Furthermore, the unknown flow on an arc is $f_{i,j}$, then the model is:

$$\begin{aligned} \min & \sum_{h, (i,j) \in A_{i,j}} c_{i,j} f_{h,i,j} \\ & \sum_{i \in A_{i,j}} f_{h,i,j} + d_{h,i,j} = \sum_{i \in A_{j,i}} f_{h,j,i} \quad \text{forall } h \in H, j \in I \\ & 0 \leq f_{h,i,j} \leq C_{i,j} \quad \text{forall } h \in H, (i,j) \in A_{i,j} \end{aligned}$$

The formulation in LPL is straightforward. Note that the flow condition exploits the sparsity of the network. This is not trivial, but necessary for larger problems.

This code is compared to the Python formulation using *Gurobi* package. Note that the *Gurobi* package also contains a *quicksum* function for faster summation than the ordinary Python *sum* operator.

LPL code (run *netflow*)

```

model netflow "Multi-commodity Flow";
  set h,commodities := ['Pencils', 'Pens'];
    i,j,nodes:=['Detroit', 'Denver',
                 'Boston', 'New York', 'Seattle'];
  parameter arcs,capacity{i,j} := [ ... ];
    cost{h,i,j} := [ ... ];
    inflow{h,j} := [ ... ];

  variable flow{h,i,j} [0..capacity];
  constraint A{h,j}: sum{i in arcs[i,j]} flow[h,i,j] +
    inflow[h,j]
      = sum{i in arcs[j,i]} flow[h,j,i];
  minimize Cost: sum{h,arcs} cost*flow;

  if GetSolverStatus()=7 then
    Write('Optimal objective %d\n', Cost);
    {h} (Write('\nOptimal flows for %s:' , h),
        Write{arcs[i,j]} ('\n  %s -> %s: %d', i,j, flow))
    ;
  end
end

```

Python/gurobipy code (download [netflow.py](#))

```

import gurobipy as gp
from gurobipy import GRB
commodities = ['Pencils', 'Pens']
nodes = ['Detroit', 'Denver',
         'Boston', 'New_York', 'Seattle']
arcs, capacity = gp.multidict({...})
cost = {...}
inflow = {...}

m = gp.Model('netflow')
flow = m.addVars(commodities, arcs, obj=cost, name="flow"
                 )
m.addConstrs(
    (flow.sum('*', i, j) <= capacity[i, j] for i, j in
     arcs), "cap")
m.addConstrs(
    (flow.sum(h, '*', j) + inflow[h, j] == flow.sum(h, j,
           '*')
     for h in commodities for j in nodes), "node")
m.optimize()

if m.Status == GRB.OPTIMAL:
    solution = m.getAttr('X', flow)
    for h in commodities:
        print('\nOptimal flows for %s:' % h)
        for i, j in arcs:

```

```

    if solution[h, i, j] > 0:
        print('%s->%s:%g' % (i, j, solution[h,
            i, j]))

```

5.3.1.2 3d Tic-Tac-Toe (3d-tic-tac-toe)

— Run LPL Code , HTML Document –

Problem: This model is the same as model 2d-tic-tac-toe, except for the dimension: it models a 3-dimensional tik-tak-toe problem.

The question here is the same as for the 2D-tik-tak-toe game: Is it possible to occupy all 27 fields without any row, column or diagonal of only **X**'s or **O**'s? (If this were not the case, then we have proven that the game cannot end in a tie.) Interestingly we can prove here that the $3 \times 3 \times 3$ tik-tak-toe cannot end in a tie (there *must* be one winner). This model is also interesting for learning some trick in MIP and logical modeling and some indexing structures.

Modeling Steps: The formulation is generalized for and 3d $n \times n \times n$ tik-tak-toe game. For the “classical” game the size is $n = 3$. The set of grid points (or fields) is then $c \in C = \{1, \dots, n^3\}$. The set of possible lines is $p \in P = \{1, \dots, 3n^2 + 6n + 4\}$ (n^2 lines in each of 3 dimensions, $6n$ 2-d diagonals and 4 inner diagonals)).

Let's identify the grid points by a horizontal and vertical position (i, j, k) with $i, j, k \in \{0, \dots, n - 1\}$. Hence, the leftmost bottom field is $(0, 0, 0)$, and the rightmost top field is $(n - 1, n - 1, n - 1)$. In the model, we also use an integer to identify a grid point: the point (i, j, k) corresponds to the integer $n^2 \cdot i + n \cdot j + k + 1$.

Now it is possible to assign the n points belonging to a line. This is done with the relation $\text{Lines}_{c,p}$ (see code). For example, the first (horizontal) line consists of the n points $1, 2, \dots, n$.

Two binary variables are used: (1) For each grid point c we want to know whether it contains a **X** or a **O** ($D_c = 1$ if it contains a **X**). (2) A binary variable for each line p is needed: $G_p = 1$ if the line p contains only **X** or only **O**.

Since all fields (cell points) must be occupied by alternately placing a **X** and a **O**, half of all points are occupied by a **X** (since the player with **X** begins, he can put 1 more than half). That is the number of all **X** on a board is $\lceil n^2 / 2 \rceil$

$$\sum_c D_c = \lceil n^2 / 2 \rceil$$

The second constraint links the D_c and the G_p variables in a logical way: for each line p , if it *does not* contain only **X** or only **O** then at least one point on

the line must be a **X** and at least must be a **O**. In other words, if $G_p = 0$ then the sum of all D_c on that line must be at least 1 and at most $n - 1$:

$$G_p \rightarrow (1 \leq \sum_{c \in \text{Lines}_{p,c}} D_c \leq n - 1) \quad \text{forall } p \in P$$

The objective is to minimize the number of G_p :

$$\min \sum_p G_p$$

The LPL formulation is a generalisation of the 3d problem. The Python implementation is from [Gurobi resources](#).

Solution: The objective is 4, which means that it is *not* possible to end the game in a tie. (Compare this result with the result in the 2D-tik-tak-toe model in [2d-tic-tac-toe](#).)

Questions

1. Solve the problem also for $n = 4$, then $n = 5$. What is the result.

Answers

1. Only change the parameter n to the corresponding value, then run. These games can end in a tie!

Further Comments: The gurobipy uses the `addGenConstrIndicator` function to specify indicator (binary) variables. LPL simply uses logical operators. Note, while in LPL the relationship `Lines` is a general implementation for any dimension n , The python code only works for $n = 3$. It uses the concept of list, while LPL uses multi-dimensional cubes to specify `Lines`. Gurobipy only works with the solver Gurobi, while LPL can send its model to any MIP solver.

LPL code (run [3d-tic-tac-toe](#))

```
model will17b "3d Tic-Tac-Toe";
parameter n:=[3]  "size of the game";
               q:=0      "q counts the numbers of lines";
set   p := 1..3*n^2+6*n+4  "set of lines";
       c := 1..n^3           "set of grid points";
Lines{p,c};
i,j,k:=0..n-1;
{i,j}(q:=q+1, {k})(Lines[q,n^2*i+n*j+k+1] := 1));
{i,k}(q:=q+1, {j})(Lines[q,n^2*i+n*j+k+1] := 1));
{j,k}(q:=q+1, {i})(Lines[q,n^2*i+n*j+k+1] := 1));
{i}  (q:=q+1, {j})(Lines[q,n^2*i+n*j+j+1] := 1));
```

```

{ i}  (q:=q+1, { j} (Lines[q,n^2*i+n*j+n-1-j+1] := 1));
{ j}  (q:=q+1, { k} (Lines[q,n^2*k+n*j+k+1] := 1));
{ j}  (q:=q+1, { k} (Lines[q,n^2*k+n*j+n-1-k+1] := 1));
{ k}  (q:=q+1, { i} (Lines[q,n^2*i+n*i+k+1] := 1));
{ k}  (q:=q+1, { i} (Lines[q,n^2*i+n*(n-1-i)+k+1] := 1))
;
q:=q+1, { i} (Lines[q,n^2*i+n*i+i+1] := 1);
q:=q+1, { i} (Lines[q,n^2*i+n*i+n-1-i+1] := 1);
q:=q+1, { i} (Lines[q,n^2*i+n*(n-1-i)+i+1] := 1);
q:=q+1, { i} (Lines[q,n^2*i+n*(n-1-i)+n-1-i+1] := 1);
binary variable
D{c}; G{p};
constraint
NoLine{p}: ~G -> (1 <= sum{c|Lines} D <= n-1);
Numb: sum{c} D = Ceil(#c/2);
minimize OBJ: sum{p} G;
Write(' %2d lines have the same color\n', OBJ);
Writep(D,G);
Write{p}(' %3d %3d \n', p, {c|Lines} c);
end

```

Python/gurobipy code (download [3d-tic-tac-toe.py](#))

```

import gurobipy as gp
from gurobipy import GRB
lines = []
size = 3

for i in range(size):
    for j in range(size):
        for k in range(size):
            if i == 0:
                lines.append(((0,j,k), (1,j,k), (2,j,k)))
            if j == 0:
                lines.append(((i,0,k), (i,1,k), (i,2,k)))
            if k == 0:
                lines.append(((i,j,0), (i,j,1), (i,j,2)))
            if i == 0 and j == 0:
                lines.append(((0,0,k), (1,1,k), (2,2,k)))
            if i == 0 and j == 2:
                lines.append(((0,2,k), (1,1,k), (2,0,k)))
            if i == 0 and k == 0:
                lines.append(((0,j,0), (1,j,1), (2,j,2)))
            if i == 0 and k == 2:
                lines.append(((0,j,2), (1,j,1), (2,j,0)))
            if j == 0 and k == 0:
                lines.append(((i,0,0), (i,1,1), (i,2,2)))
            if j == 0 and k == 2:
                lines.append(((i,0,2), (i,1,1), (i,2,0)))
lines.append(((0,0,0), (1,1,1), (2,2,2)))

```

```

lines.append(((2,0,0), (1,1,1), (0,2,2)))
lines.append(((0,2,0), (1,1,1), (2,0,2)))
lines.append(((0,0,2), (1,1,1), (2,2,0)))

model = gp.Model('Tic_Tac_Toe')
isX = model.addVars(size, size, size, vtype=GRB.BINARY,
                     name="isX")

```

... continue with Python code

```

isLine = model.addVars(lines, vtype=GRB.BINARY, name="isLine")
x14 = model.addConstr(isX.sum() == 14)
for line in lines:
    model.addGenConstrIndicator(isLine[line], False, isX[
        line[0]] + isX[line[1]] + isX[line[2]] >= 1)
    model.addGenConstrIndicator(isLine[line], False, isX[
        line[0]] + isX[line[1]] + isX[line[2]] <= 2)

model.setObjective(isLine.sum())
model.optimize()

```

5.3.1.3 Non-negative Regression (regression1)

— Run LPL Code , HTML Document –

Problem: Consider the following regression problem with $A \in \mathbf{R}^{m,n}$, $b \in \mathbf{R}^m$ (typically $m \geq n$). We want to solve:

$$\min_{x \geq 0} \|b - Ax\|_2^2$$

The problem is from [Regression Model](#).

Modeling Steps: The optimization problem can be formulated (in matrix form) as follows:

$$\begin{aligned}
 \min_{x,r} \quad & r^T r \\
 \text{s.t.} \quad & r = b - Ax \\
 & x \geq 0 \\
 & x \in \mathbf{R}^n \quad (\text{regression weights}) \\
 & r \in \mathbf{R}^m \quad (\text{residual}) \\
 & b \in \mathbf{R}^m \quad (\text{data})
 \end{aligned}$$

Further Comments: In the newest version of *gurobipy* (10.0 – 2022) one can use MVar, MLinExpr, MQuadExpr to model variables and constraints that are naturally expressed in “vectorized” form. This often allow to write compact, loopless code and most arithmetic operations allow for *ndarray* (numpy package) and *scipy.sparse* operands.

Frankly, I do not find that this more compact form is a big deal. First of all, sparse datacubes – which is the rule and not the exception – must still be generated using loops, and only simple model can use this vectorized form. Second, one must be careful in using nested @ operators: depending on the nesting the generation of the model can be more or less efficient.

Despite of Python’s vectorized form, the LPL code is still more concise. Note that in LPL the passive can be omitted, this make the syntax even more “vector-like”. If one likes to generate exactly the same model instance in LPL then the data can be read from the data file `regressionAb.txt` (`regressionAb.txt`).

LPL code (run `regression1`)

```
model regression "Non-negative Regression";
parameter n:=10; m:=100;
set i:=1..m;
j:=1..n;
parameter A{i,j}:=Rnd(0,1);
b{i}:=Rnd(0,1);
variable x{j};
r{i} [-1..1];
constraint C0{i}: r = b - sum{j} A*x;
minimize Dev: sum{i} r*r;
Write('x = %9.6f\n', {j} x);
end
```

Python/gurobipy code (download `regression1.py`)

```
import math
import numpy as np
import scipy.sparse as sp
import gurobipy as gp
import pandas as pd
gp.setParam('OutputFlag',0)

# Example data
A = np.random.rand(100, 10)
b = A \verb?+? np.random.rand(10) + 0.01 * np.random.rand(100)

model = gp.Model()
x = model.addMVar(10, name="x")
r = model.addMVar(100, lb=-np.inf, name="r")
model.setObjective(r ? r)
model.addConstr(r == b - A \verb?+? x)
```

```

model.optimize()
print(f"Regression_values:{x.X}")
print(f"Fit_error:{math.sqrt(r.X?r.X):.2e}")

```

5.3.2 OR-tools (Google)

OR-Tools is an open source software suite for optimization, to formulate and solve vehicle routing, flows, integer and linear programming, and constraint programming problems. It includes the open-source solvers *SCIP*, *GLPK*, or Google's *GLOP* and award-winning *CP-SAT* and can be linked to *Gurobi* and *Cplex*. The models can be implemented using a package in C++, Python, C#, or Java. The tools is install using the following command:

```
python -m pip install --upgrade --user ortools
```

5.3.2.1 A Simple Nurse Scheduling (nurses)

— Run LPL Code , HTML Document –

Problem: Problem: Find an optimal assignment of nurses to shifts (tasks) for a duration of 7 days. Each nurse can request to be assigned to specific shifts. Each shift can only be assigned to exactly one nurse. In a day, a nurse works at most in one shift. Furthermore, each nurse should work the same number of shifts than another nurse in the whole period if possible. The optimal assignment maximizes the number of fulfilled shift requests (the problem is from **OR-Tools**).

Modeling Steps: Let's introduce three sets: (1) for the nurses: $n \in N = \{1, \dots, N\}$, (2) for the periods (days): $d \in D = \{1, \dots, D\}$, and for the shifts (tasks): $s \in S = \{1, \dots, S\}$. A Boolean relationship for the shift requests is introduced as $req_{n,d,s}$ which is true if nurse n made a request to be assigned to shift s in day d . A binary variable $x_{n,d,s}$ ("shifts" in the code) defines whether nurse n works on shift s in day d ($=1$) or not ($=0$). The parameter $minS$ is a minimal number of shifts that a nurse must work at least to fullfill all assignments. $maxS$ is just at most one bigger than $minS$.

The whole model is:

$$\begin{aligned} \max & \sum_{\substack{n,d,s | \text{req}_{n,d,s}}} x_{n,d,s} \\ \text{s.t.} & \sum_n x_{n,d,s} = 1 \quad \text{forall } d \in D, s \in S \quad (1) \\ & \sum_s x_{n,d,s} \leq 1 \quad \text{forall } n \in N, d \in D \quad (2) \\ & \min S \leq \sum_{d,s} x_{n,d,s} \leq \max S \quad \text{forall } n \in N \quad (3) \end{aligned}$$

Constraint (1) specifies that for each shift in all days exactly one nurse is assigned. Constraint (2) determines that each nurse works at most one shift per day. Constraint (3) distribute evenly the amount of work (all shifts: $D \cdot S$) to every nurse. On average each nurse must work $D \cdot S/N$. If this is not an integer then some nurses must work $\lfloor D \cdot S/N \rfloor$ and some must work an additional shift. Hence, every nurse must work in at least $\min S$ shifts and at most $\max S$ shifts.

LPL code (run **nurses**)

```

model nurses "A Simple Nurse Scheduling";
parameter N, num_nurses := 5;
           D, num_days := 7;
           S, num_shifts := 3;
set n,all_nurses := 1..N;
set d,all_days := 1..D;
set s,all_shifts := 1..S;
set req{n,d,s} "shift requests";
Read{n,d,s}('nurses.txt',;1',n,d,s,req);
parameter minS := Trunc((S*D)/N);
maxS := minS + if(S*D%N,1);
binary variable shifts{n,d,s};
constraint C1{d,s}: sum{n} shifts = 1;
C2{n,d}: sum{s} shifts <= 1;
C3{n}: minS <= sum{d,s} shifts <= maxS;
maximize Req: sum{req[n,d,s]} shifts;
Write{d}('Day %d\n%', n,
Format{n,s|shifts}(' Nurse %d works in shift %s %s\
', n,
s, if(req,'(requested)', '')),
Format{n,s~shifts and req}(' Nurse %d requested
shift %s but does not work\n', n,
s);
Write('The number of requests fulfilled: %d out of %d\n
',Req,sum{req})1);
end
```

The data file is (see **nurses.txt**):

```

5 7 3 //number nurses/days/shifts
// the request list is:
1 1 3 //nurse 1 requests in day 1 shift 3
1 5 3
....

```

Python/or-tools code ([download nurses.py](#))

```

from ortools.sat.python import cp_model
def main():
    num_nurses = 5
    num_shifts = 3
    num_days = 7
    all_nurses = range(num_nurses)
    all_shifts = range(num_shifts)
    all_days = range(num_days)
    shift_requests =
        [[[0,0,1], [0,0,0], [0,0,0], [0,0,0], [0,0,1],
          [0,1,0], ...]]
    model = cp_model.CpModel()
    shifts = {}                      # declare variables
    for n in all_nurses:
        for d in all_days:
            for s in all_shifts:
                shifts[(n, d,
                           s)] = model.NewBoolVar('shift_n%id%is%' % (n
                           , d, s))
    for d in all_days:                 # constraint 1
        for s in all_shifts:
            model.AddExactlyOne(shifts[(n, d, s)] for n in
                                   all_nurses)
    for n in all_nurses:               # constraint 2
        for d in all_days:
            model.AddAtMostOne(shifts[(n, d, s)] for s in
                                   all_shifts)
    min_shifts_per_nurse = (num_shifts * num_days) //
                           num_nurses
    if num_shifts * num_days % num_nurses == 0:
        max_shifts_per_nurse = min_shifts_per_nurse
    else:
        max_shifts_per_nurse = min_shifts_per_nurse + 1

```

continuing with Python code ...

```

for n in all_nurses:                  # constraint 3
    num_shifts_worked = 0
    for d in all_days:
        for s in all_shifts:
            num_shifts_worked += shifts[(n, d, s)]

```

```

model.Add(min_shifts_per_nurse <= num_shifts_worked
        )
model.Add(num_shifts_worked <= max_shifts_per_nurse
        )
model.Maximize(
    sum(shift_requests[n][d][s] * shifts[(n, d, s)] for
        n in all_nurses
        for d in all_days for s in all_shifts))
solver = cp_model.CpSolver()
status = solver.Solve(model)

if status == cp_model.OPTIMAL: # output
    print('Solution:')
    for d in all_days:
        print('Day', d)
        for n in all_nurses:
            for s in all_shifts:
                if solver.Value(shifts[(n, d, s)]) == 1:
                    if shift_requests[n][d][s] == 1:
                        print('Nurse', n, 'works_shift', s, '(requested).')
                    else:
                        print('Nurse', n, 'works_shift', s,
                            '(not_requested).')
                print()
    print(f'Number_of_shift_requests_met={solver.
        ObjectiveValue()}', f'(out_of_{num_nurses}*'
        min_shifts_per_nurse))')
else:
    print('No_optimal_solution_found!')

if __name__ == '__main__':
    main()

```

5.3.3 Pyomo (Python – open source)

Pyomo uses object-oriented capacities of Python to represent optimization models by simulating the components of a AML (algebraic modeling language) (see homepage: [Pyomo](#). A new edition of the book has been published recently (2021) from the authors “Pyomo is a flexible, extensible modeling framework that captures and extends central ideas found in modern algebraic modeling languages, all within the context of a widely used programming language.” [7]. It allows the user to formulate a large range of models: linear, quadratic, nonlinear, mixed integer, disjunctive, dynamic problems with differential equations, and models with equilibrium constraints. It can be linked

to an number of solvers and though its output of the AMPL nl format to all their solvers. A collection of interesting models commposed by Alireza Soroudi can also be found at [Pyomo Models](#).

A Pyomo model is implemented as a Python object *model*, which contains a collection of *components*, simulating the components of an algebraic modeling language:

Set	defines set data (index-sets)
Param	defines data parameters
Var	defines variables
Constraint	defines constraints
Objective	defines objectives

One must choose whether to define a *concrete* or an *abstract* model. Concrete models include the data directly in the model code, abstract models separate strictly data from the model structure – in order that the structure can be reused. In this case, the data are external and “linked” through a similar mechanism as AMPL though its `data` command.

Pyomo can also formulate nonlinear problems. Nonlinear solver generally require evaluation of first, and often second, derivatives. Pyomo uses automatic differentiation (AD) tools to provide them. If connected to a AMPL solver, this task is delegated to the AMPL Solver Library (ASL) interface (which is open source). A small example is given that minimizes the Rosenbrock function.

An example of a concrete and an abstract model – comparing to LPL – is given below (from [7]).

5.3.3.1 Warehouse Location ([wl1](#))

— Run LPL Code , [HTML Document](#) –

Problem: This model represents a location problem: a set of customers has to be served from a potential set of warehouses. The number of warehouses to open is limited and the delivery costs from a warehouse to a costumer location is given. The question is which warehouse to open and what quantity to deliver to each customer.

Modeling Steps: The set of customers is $m \in M$ and the potential set of warehouses is $n \in N$. The number of warehouses to open is P and the delivery costs from a warehouse n to a costumer location m is $d_{n,m}$. Let $x_{n,m}$ be the unknown quantity send from warehouse n to customer m and let y_n be a binary variable which is 1 if the warehouse location is open.

The mathematical model can be formulated as follows:

$$\begin{aligned}
 \text{min} \quad & \sum_{n,m} c_{n,m} x_{n,m} \\
 \text{subject to} \quad & \sum_n x_{n,m} = 1 \quad \text{forall } m \in M \\
 & x_{n,m} \leq y_n \quad \text{forall } n \in N, m \in M \\
 & \sum_n y_n = P \quad \text{forall } n \in N \\
 & 0 \leq x_{n,m} \leq 1 \quad \text{forall } n \in N, m \in M \\
 & y_n \in \{0, 1\} \quad \text{forall } n \in N
 \end{aligned}$$

This model is used to compare LPL with Pyomo. See for an implementation of an abstract model here: [wl2](#).

Further Comments: The model in LPL includes all data directly into the model. The Pyomo Python script formulation below uses plain Python code for the data implementation. The model specification begins by creating a concrete model with:

```
model = pyo.ConcreteModel(name="(WL)")
```

It is followed by the declaration of variables, objective and constraints. Next the connection to the solver is established:

```
solver = pyo.SolverFactory('gurobi')
```

Then the solver is called, and finally the output is generated with the `print` and with Pyomo's `pprint` function.

LPL code (run [wl1](#))

```

model wl1 "Warehouse Location";
set n := [Harlingen Memphis Ashland] "a set of
warehouses";
m := [NYC LA Chicago Houton] "a set of customer
locations";
parameter P := 2 "number of open warehouses";
d{n,m} "delivery costs"
:= [1956 1606 1410 330
   1096 1792 531 567
   485 2322 324 1236];
variable x{n,m} [0..1] "fraction of demand";
binary y{n} "warehouse open?";
constraint A{m}: sum{n} x[n,m] = 1;
B{n,m}: x[n,m] <= y[n];
C: sum{n} y[n] = P;
minimize obj: sum{n,m} d[n,m]*x[n,m];

```

```

    Write('OBJ=%5.2f\n',obj);
    Write('Open: %9s\n', {n|y} n);
end

```

Python/gurobipy code ([download wl1.py](#))

```

import pyomo.environ as pyo
import gurobipy

# the data
N = ['Harlingen', 'Memphis', 'Ashland']
M = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956, \
      ('Harlingen', 'LA'): 1606, \
      ('Harlingen', 'Chicago'): 1410, \
      ('Harlingen', 'Houston'): 330, \
      ('Memphis', 'NYC'): 1096, \
      ('Memphis', 'LA'): 1792, \
      ('Memphis', 'Chicago'): 531, \
      ('Memphis', 'Houston'): 567, \
      ('Ashland', 'NYC'): 485, \
      ('Ashland', 'LA'): 2322, \
      ('Ashland', 'Chicago'): 324, \
      ('Ashland', 'Houston'): 1236 }
P = 2

# the model
model = pyo.ConcreteModel(name="(WL)")
model.x = pyo.Var(N, M, bounds=(0,1))
model.y = pyo.Var(N, within=pyo.Binary)

def obj_rule(mdl):
    return sum(d[n,m]*mdl.x[n,m] for n in N for m in M)
model.obj = pyo.Objective(rule=obj_rule)

def demand_rule(mdl, m):
    return sum(mdl.x[n,m] for n in N) == 1
model.demand = pyo.Constraint(M, rule=demand_rule)

```

... continue with Python code

```

def warehouse_active_rule(mdl, n, m):
    return mdl.x[n,m] <= mdl.y[n]
model.warehouse_active = pyo.Constraint(N, M, rule=
    warehouse_active_rule)

def num_warehouses_rule(mdl):
    return sum(mdl.y[n] for n in N) <= P

```

```

model.num_warehouses = pyo.Constraint(rule=
    num_warehouses_rule)

# the solver interface
solver = pyo.SolverFactory('gurobi')
res = solver.solve(model)
pyo.assert_optimal_termination(res)

print("OBJ=",pyo.value(model.obj))
model.y pprint()
model.x pprint()

```

5.3.3.2 Warehouse Location ([wl2](#))

— Run LPL Code , [HTML Document](#) –

Problem: This model [wl2](#) is the same model as model [wl1](#), with the exception that the data set is separated from the model structure.

To solve the problem, call the LPL model in the command line as follows:

```
lplc wl2 - @INF='wl2-data.dat' #@OUTF='wl2-out.dat'
```

The file `wl2-data.dat` contains two data sets. If you want to run the second set `data1` then call it as follows:

```
lplc wl2 - @INF='wl2-data.dat'#
            @OUTF='wl2-out.dat'#@IN='data1'
```

Further Comments: The model in LPL contains only the model structure without any data. The data are defined in an external file (see code).

The Pyomo Python script formulation below begins by creating an abstract model with:

```
model = pyo.AbstractModel(name="(WL)")
```

Like the concrete model, it defines the data parameters, the variables, the constraints, and the objective function. It can be executed but it does nothing, since the data are missing and solver link is not established.

The Pyomo abstract model is as follows:

While installing Pyomo with the pip command, an executable `pyomo.exe` is also installed. To run and solve the model, use this executable and run the script on the command line as follows:

```
pyomo --solver=gurobi wl2.py wl2-pydata.dat
```

LPL code (run **wl2**)

```
model wl2 "Warehouse Location";
set n "a set of warehouses";
m "a set of customer locations";
parameter P "number od open warehouses";
d{n,m} "delivery costs";
variable x{n,m} [0..1] "fraction of demand";
binary y{n} "warehouse open?";
constraint A{m}: sum{n} x[n,m] = 1;
B{n,m}: x[n,m] <= y[n];
C: sum{n} y[n] = P;
minimize obj: sum{n,m} d[n,m]*x[n,m];
end
```

The data are defined in an external file **wl2-data.dat** that must be linked when called LPL (see above) :

```
model data;
n:= [Harlingen Memphis Ashland];
m:= [NYC LA Chicago Houton];
P:=2;
d{n,m} := [1956 1606 1410 330
           1096 1792 531 567
           485 2322 324 1236];
end
```

The solution output instructions are also in a separate file **wl2-out.dat**):

```
model output;
Write('OBJ=%5.2f\n',obj);
Write('Open: %9s\n', {n|y} n);
end
```

Promo/gurobipy code (download **wl2.py**)

```
# wl2.py: AbstractModel version of warehouse location
import pyomo.environ as pyo

model = pyo.AbstractModel(name="(WL)")
model.N = pyo.Set()
model.M = pyo.Set()
model.d = pyo.Param(model.N,model.M)
model.P = pyo.Param()
model.x = pyo.Var(model.N, model.M, bounds=(0,1))
model.y = pyo.Var(model.N, within=pyo.Binary)

def obj_rule(model):
```

```

return sum(model.d[n,m]*model.x[n,m] for n in model.N
           for m in model.M)
model.obj = pyo.Objective(rule=obj_rule)

def one_per_cust_rule(model, m):
    return sum(model.x[n,m] for n in model.N) == 1
model.one_per_cust = pyo.Constraint(model.M, rule=
    one_per_cust_rule)

def warehouse_active_rule(model, n, m):
    return model.x[n,m] <= model.y[n]
model.warehouse_active = pyo.Constraint(model.N, model.M,
    rule=warehouse_active_rule)

def num_warehouses_rule(model):
    return sum(model.y[n] for n in model.N) <= model.P
model.num_warehouses = pyo.Constraint(rule=
    num_warehouses_rule)

```

An external file **wl2-pydata.dat** holds the data (the format is similar to the AMPL data format):

```

# wl2_data.dat: Pyomo format data file for the warehouse
location problem
set N := Harlingen Memphis Ashland ;
set M := NYC LA Chicago Houston;
param d :=
    Harlingen NYC 1956
    Harlingen LA 1606
    Harlingen Chicago 1410
    Harlingen Houston 330
    Memphis NYC 1096
    Memphis LA 1792
    Memphis Chicago 531
    Memphis Houston 567
    Ashland NYC 485
    Ashland LA 2322
    Ashland Chicago 324
    Ashland Houston 1236
;
param P := 2 ;

```

5.3.3.3 Rosenbrock Function (**rosenbrock**)

— Run LPL Code , HTML Document –

Problem: The Rosenbrock function is the non-linear function:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

This function is to be minimized.

Modeling Steps: The model in LPL is straightforward (the solver used is Knitro as defined in the lplcfg.lpl file). It is compared to the Pyomo Python script formulation of a concrete model (note that the free source ipopt is used to solve the model).

LPL code (run `rosenbrock`)

```
model w1 "Rosenbrock Function";
variable x:=1.5; y:=1.5;
minimize obj: (1-x)^2 + 100*(y-x^2)^2;
Write('OBJ=%5.5f\n',obj);
Write('x=%5.5f , y=%5.5f\n', x,y);
end
```

Python/Promo code (download `rosenbrock.py`)

```
import pyomo.environ as pyo
model = pyo.ConcreteModel()
model.x = pyo.Var(initialize=1.5)
model.y = pyo.Var(initialize=1.5)
def rosenbrock(model):
    return (1.0 - model.x)**2 \
        + 100.0*(model.y - model.x**2)**2
model.obj = pyo.Objective(rule=rosenbrock, sense=pyo.
    minimize)
status = pyo.SolverFactory('ipopt').solve(model)
pyo.assert_optimal_termination(status)
model.pprint()
```

Pyomo can also represent hierarchically structured problems using the component `Block` to manage model scope. Various extensions for disjunctive programming (component `Disjunction`), differential algebraic equations (DAE) (components `ContinuousSet`, `DerivativeVar`), models with equilibrium constraints (component `Complementarity`) exist. These extensions can be automatically transformed into an appropriate form to be solved. For example, DAEs can be transformed into a general nonlinear model using, p.e. the Euler forward method.

5.3.4 Gecco/APMonitor (Python – open source)

APMonitor is an algebraic modeling language for optimization models of

mixed-integer and differential algebraic equations. It is coupled with large-scale solvers for linear, quadratic, nonlinear, and mixed integer programming (LP, QP, NLP, MILP, MINLP). Modes of operation include data reconciliation, real-time optimization, dynamic simulation, and nonlinear predictive control. It is freely available through MATLAB, Python, and from a web browser interface.

Gekko is a Python package and a front-end interface to APMonitor. Gekko is embedded in Python as a object initialized in Python with :

```
m = GEKKO()
```

Variables, Equations, solve instructions can then be defined simply as (see example below):

```
m.Var(...)  
m.Equation(...)  
m.solve(...)
```

To install GEKKO as a Python package enter the command on the terminal:

```
pip install gekko
```

There is no need to install a solver since gekko communicates with a server at APMonitor to solve the problem. In the following, two examples are given which compare APMonitor/gekko with LPL.

5.3.4.1 A Hock/Schittkowski Model ([hs71](#))

— [Run LPL Code](#) , [HTML Document](#) –

Problem: A non-linear benchmark model (NLP) from the Hock & Schittkowski model library with 4 variables and 2 constraints.

Modeling Steps: A mathematical modeling of this problem is as follows :

$$\begin{aligned} \min \quad & x_1 x_4 (x_1 + x_2 + x_3) + x_3 \\ \text{s.t.} \quad & x_1 x_2 x_3 x_4 \geq 25 \\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40 \\ & 1 \leq x_1, x_2, x_3, x_4 \leq 5 \\ & x_0 = (1, 5, 5, 1) \quad \text{initialization} \end{aligned}$$

LPL code (run [hs71](#))

```

model hs71 "A Hock/Schittkowski model";
  set i := [1..4];
  variable x{i} [1..5] := [1 5 5 1];
  constraint
    A: prod{i} x[i] >= 25;
    B: sum{i} x^2 = 40;
  minimize O: x[1]*x[4]*(x[1]+x[2]+x[3]) + x[3];
  Write('x1: %5.3f\nx2: %5.3f\nx3: %5.3f\nx4: %5.3f\n',x
    [1],x[2],x[3],x[4]);
  Write('Objective: %5.3f\n',O);
end

```

The AMP model (right side) can be solved online or using the following Python script:

Python script (run **hs71-main.py**)

```

import sys
sys.path.append("../")
from apm import *

# Solve optimization problem
sol = apm_solve('hs71',3)

print('---_Results_of_the_Optimization_Problem---')
print('x[1]:' + str(sol['x[1]']))
print('x[2]:' + str(sol['x[2]']))
print('x[3]:' + str(sol['x[3]']))
print('x[4]:' + str(sol['x[4]']))

```

APMonitor code (run at **hs071.apm**)

```

Model hs71
Variables
  x1 = 1, >=1, <=5
  x2 = 5, >=1, <=5
  x3 = 5, >=1, <=5
  x4 = 1, >=1, <=5
End Variables
Equations
  x1*x2*x3*x4 > 25
  x1^2 + x2^2 + x3^2 + x4^2 = 40
  ! best known objective = 17.0140173

```

```

    minimize x1*x4*(x1+x2+x3) + x3
End Equations
End Model

```

Python/GEKKO code ([download hs71.py](#))

```

from gekko import GEKKO
m = GEKKO() # Initialize gekko
# Use IPOPT solver (default)
m.options.SOLVER = 3
# Change to parallel linear solver
m.solver_options = ['linear_solver_ma97']
# Initialize variables
x1 = m.Var(value=1,lb=1,ub=5)
x2 = m.Var(value=5,lb=1,ub=5)
x3 = m.Var(value=5,lb=1,ub=5)
x4 = m.Var(value=1,lb=1,ub=5)
# Equations
m.Equation(x1*x2*x3*x4>=25)
m.Equation(x1**2+x2**2+x3**2+x4**2==40)
m.Obj(x1*x4*(x1+x2+x3)+x3) # Objective
m.options.IMODE = 3 # Steady state optimization
m.solve(disp=False) # Solve

```

... continue with Python script

```

print('Results')
print('x1:' + str(x1.value))
print('x2:' + str(x2.value))
print('x3:' + str(x3.value))
print('x4:' + str(x4.value))
print('Objective:' + str(m.options.objfcnval))

```

5.3.4.2 Problem 8 from APMonitor ([prob8](#))

— Run LPL Code , HTML Document –

Problem: Implement the following differential equation with initial condition $y(0) = 5$.

$$K \frac{dy}{dt} = -ty \quad (1)$$

where $K = 10$. The solution should be reported from initial time 0 to final time 20 (model is from: [GEKKO](#)).

Modeling Steps: In LPL one needs to discretize the differential equations manually. Using a time step Δt of 0.25, we get $n = 80$ time steps till 20. Let's define a set $i \in I = \{0, \dots, n\}$, and a data vector of time points $t_i = i/4$.

This problem – a simple ODE (ordinary differential equation) – can now be approximate by (Euler forward method) :

$$\frac{y_{i+1} - y_i}{\Delta t} = -\frac{-t_i y_i}{K} \quad \text{forall } i \in I - \{n\}$$

or

$$y_{i+1} = y_i + \frac{\delta t}{K} \cdot -t_i y_i \quad \text{forall } i \in I - \{n\} \quad \text{with } y_0 = 5$$

This can be implemented using a loop: starting with y_0 – which is given – we calculate y_1 , then y_2 , etc. (see code below).

The Euler forward method is only a rough approximation. Better approximations are the Runde-Kutta methods. As an example, see the Runde-Kutta second order method as follows (see for example [44] and [2]).

$$\begin{aligned} k_1 &= \Delta t \cdot (-t_i y_i) / K \\ k_2 &= \Delta t \cdot ((-t_i y_i) + k_1 / 2) / K \\ y_{i+1} &= y_i + k_2 \end{aligned}$$

In LPL, these methods can be implemented using a loop and the graph can be generated by the `Draw` instruction.

Another approach is to transform the differential equations into ordinary (non)-linear algebraic equations and to solve it with a NLP solver [44]. The model `prob8a` in LPL formulation is an example on how the differential equation (1) can be transformed into an algebraic equation (using Euler's forward method)⁵.

Further Comments: The dollar sign \$ in the APMonitor code indicates a derivative. The model is transformed using orthogonal collocation into a NLP (non-linear) model and solved by the open source APopt or IPopt solvers. In LPL, this transformation must be done manually.

LPL code (run `prob8`)

```
| model prob8 "Problem 8 from APMonitor";
|   set i:=0..80;
|   parameter D:=0.25  "step size";
```

⁵Warning: This approach is only for educational demonstration. It should not be used in a professional context, because the errors can grow rapidly. Most software solving such problem use orthogonal collocation and other methods, see [17].

```

parameter t{i}:=i*D;
parameter y{i};;
y[0]:=5;
-- Euler's forward method
{i|i<#i} (y[i+1] := (y[i] + D*(-t*y)/10));
--Runde-Kutta second order method:
//parameter k1; k2;
//{i|i<#i} (k1:=D*(-t*y)/10,
//           k2:=D*((-t*y)+k1/2)/10,
//           y[i+1]:=y + k2);
Draw.Scale(1,1);
Draw.XY(t,y,1);
end

```

Alternative implementation (run prob8a)

```

model prob8a "Problem 8 from APMonitor";
  set i:=0..80;
  parameter D:=0.25 "step size";
  parameter t{i}:=i*D;
  variable y{i};
  constraint
    B: y[1] = 5;
    A{i|i<#i}: y[i+1] = y[i] + 0.25*(-t*y)/10;
  solve;
  Draw.Scale(1,1);
  Draw.XY(t,y,1);
end

```

APMonitor code (download prob8.apm)

```

Model prob8
  Parameters
    k = 10
  End Parameters
  Variables
    t > 0 < 20
    y = 5 ! initial condition
  End Variables
  Equations
    k * $y = -t*y
  End Equations
End Model

```

Python/GEKKO code (download prob8.py)

```

from gekko import GEKKO
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline ## use this only in Jupyter

```

```

m = GEKKO()
m.time = np.linspace(0,20,100)
k = 10
y = m.Var(value=5.0)
t = m.Param(value=m.time)
m.Equation(k*y.dt() == -t*y)
m.options.IMODE = 4
m.solve(disp=False)

plt.plot(m.time,y.value)
plt.xlabel('time')
plt.ylabel('y')
plt.savefig('prob8.png') ## save to a file

```

5.3.5 PuLP (Python – open source)

Pulp is an open source package that can formulate linear MIP models. It can be linked to a numbers of linear solvers (CBC, GLPK, CLPLEX, Gurobi, etc.). To install the PuLP use pip:

```
pip install pulp
```

To use the package a model must be initialized with :

```
prob = LpProblem(...)
```

Variables are declared using:

```
x = LpVariable(...)
choices = LpVariable.dicts(...) ## an indexed variable
```

The constraints are added by adding expression to the list:

```
prob += lpSum([choices[v][r][c] for c in Cols]) == 1,""
```

Below the Sudoku problem is used to illustrate the syntax of PuLP.

5	3		7					
6			1	9	5			
9	8					6		
8			6					3
4		8		3				1
7			2					6
6				2	8			
	4	1	9					5
		8				7	9	

Figure 5.5: A Sudoku Instance

5.3.5.1 A Sudoku Instance ([sudokuP](#))

— Run LPL Code , [HTML Document](#) —

Problem: The Sudoku is a well known game. Its rules are simple: “Fill in the grid (see, for example, Figure 5.5) so that every row, every column, and every 3×3 subblock contains the digits 1 to 9.”. The modeling has been explained in my [Puzzlebook](#). An implementation can be found at [sudoku](#).

Modeling Steps: A mathematical modeling of this problem is as follows :

$$\begin{aligned}
 \sum_k x_{i,j,k} &= 1 \quad \text{forall } i, j \in I \\
 \sum_j x_{i,j,k} &= 1 \quad \text{forall } i, k \in I \\
 \sum_i x_{i,j,k} &= 1 \quad \text{forall } j, k \in I \\
 \sum_j x_{i',j',k} &= 1 \quad \text{forall } i, k \in I \\
 &\qquad\qquad\qquad \text{with } i' = i - (i-1) \bmod 3 + \lceil (j-1)/3 \rceil \\
 &\qquad\qquad\qquad \text{and } j' = 3(i-1) \bmod 3 + (j-1) \bmod 3 + 1 \\
 x_{i,j,k} &= 1 \quad \text{forall } i, j, k \in I, P_{i,j} = k \\
 x_{i,j,k} &\in \{0, 1\} \quad \text{forall } i, j, k \in I \\
 I &= \{1, \dots, 9\}
 \end{aligned}$$

The first constraint make sure that in every cell exactlyone digit is placed. The second and thrid constraints garantuees that on each row and column every digit occurs exactly once, and the fourth constraint requires that in each 3×3 sub-block every digit occurs. Finally, the given entries must be set.

Further Comments: The script uses a default solver, if we want to use a different solver, we can modify the script as follows:

```
import pulp as pl
path_to_cplex = r'c:\....\cplex.exe'
solver = pl.CPLEX_CMD(path=path_to_cplex)
...
result = model.solve(solver)
```

Note the concise form of writing the output in LPL.

LPL code (run **sudokuP**)

```
model sudoku "A Sudoku Instance";
set i,j,k := 1..9;
parameter P{i,j} := [5 3 0 0 7 0 0 0 0
                     6 0 0 1 9 5 0 0 0
                     0 9 8 0 0 0 0 6 0
                     8 0 0 0 6 0 0 0 3
                     4 0 0 8 0 3 0 0 1
                     7 0 0 0 2 0 0 0 6
                     0 6 0 0 0 0 2 8 0
                     0 0 0 4 1 9 0 0 5
                     0 0 0 0 8 0 0 7 9];
binary variable x{i,j,k} "k is in cell (i,j)?";
constraint
  N{i,j}: sum{k} x = 1 "In each cell only one k";
  R{i,k}: sum{j} x = 1 "Every k in each row i";
  C{j,k}: sum{i} x = 1 "Every k in each column j";
  B{i,k}: sum{j} //subblocks
    x[i-(i-1)%3+Trunc((j-1)/3), 3*(i-1)%3+(j-1)%3+1, k]
    = 1;
  F{i,j,k|P[i,j]=k}: x = 1 ;
solve;
integer parameter X{i,j} := argmax{k} x;
Write('+-----+-----+-----+\n');
Write{i}(' | %s| %s| %s|\n%', 
  Format{j|j<=3}('%1d ',X),
  Format{j|4<=j<=6}('%1d ',X),
  Format{j|j>=7}('%1d ',X),
  if(i=3 or i=6 or i=9,
    Format(' +-----+-----+-----+\n'), ''));
end
```

Python/PuLP code (download [sudokuP.py](#))

```
# Authors: Antony Phillips, Dr Stuart Mitcheall
from pulp import *
Sequence = ["1", "2", "3", "4", "5", "6", "7", "8", "9"]
Vals = Sequence
Rows = Sequence
Cols = Sequence
Boxes = []
for i in range(3):
    for j in range(3):
        Boxes += [ [(Rows[3*i+k], Cols[3*j+l]) for k in range
                    (3) for l in range(3)]]

# the model
prob = LpProblem("Sudoku_Problem", LpMinimize)
choices = LpVariable.dicts("Choice", (Vals, Rows, Cols), 0, 1,
                           LpInteger)
prob += 0, "Arbitrary_Objective_Function"
for r in Rows:
    for c in Cols:
        prob += lpSum([choices[v][r][c] for v in Vals]) == 1,
                 ""
for v in Vals:
    for r in Rows:
        prob += lpSum([choices[v][r][c] for c in Cols]) == 1,
                 ""
    for c in Cols:
        prob += lpSum([choices[v][r][c] for r in Rows]) == 1,
                 ""
    for b in Boxes:
        prob += lpSum([choices[v][r][c] for (r,c) in b]) ==
                1, ""
prob += choices["5"]["1"]["1"] == 1, ""
prob += choices["6"]["2"]["1"] == 1, ""
prob += choices["8"]["4"]["1"] == 1, "
```

The same output of LPL and PuLP is as follows:

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6

```

| 9 6 1 | 5 3 7 | 2 8 4 |
| 2 8 7 | 4 1 9 | 6 3 5 |
| 3 4 5 | 2 8 6 | 1 7 9 |
+-----+-----+-----+

```

... continue with Python code

```

prob += choices["4"]["5"]["1"] == 1, ""
prob += choices["7"]["6"]["1"] == 1, ""
prob += choices["3"]["1"]["2"] == 1, ""
prob += choices["9"]["3"]["2"] == 1, ""
prob += choices["6"]["7"]["2"] == 1, ""
prob += choices["8"]["3"]["3"] == 1, ""
prob += choices["1"]["2"]["4"] == 1, ""
prob += choices["8"]["5"]["4"] == 1, ""
... some lines have been cut ...
prob += choices["6"]["6"]["9"] == 1, ""
prob += choices["5"]["8"]["9"] == 1, ""
prob.writeLP("Sudoku.lp")
prob.solve()

# the output
sudokuout = open('sudokuout.txt', 'w')
for r in Rows:
    if r == "1" or r == "4" or r == "7":
        sudokuout.write("-----+\n")
    for c in Cols:
        for v in Vals:
            if value(choices[v][r][c]) == 1:
                if c == "1" or c == "4" or c == "7":
                    sudokuout.write("|_ ")
                sudokuout.write(v + "_")
                if c == "9":
                    sudokuout.write("|\n")
sudokuout.write("-----+")
sudokuout.close()
print ("Status:", LpStatus[prob.status])
print ("Solution Written to sudokuout.txt")

```

5.3.6 Python-MIP (Python – open source)

Python-MIP is another high-performance collection of Python tools for the modeling and solution of Mixed-Integer Linear programs (MIPs). Its syntax was inspired by Pulp, but the package also provides access to advanced solver features like cut generation, lazy constraints, MIP starts, and solution pools. We use the *queens* problem on a chessboard to give a glimpse of its syntax.

5.3.6.1 The n-Queens Problem ([queens](#))

— [Run LPL Code](#) , [HTML Document](#) —

Problem: On a $n \times n$ chessboard place as many queens as possible which to not beat each other. (Other formulations of this problem are explained in my puzzlebook found at [my shop](#)). The Python code is from [Python-MIP](#)

Modeling Steps: Let n be the size of the chessboard and $i, j \in I = \{1, \dots, n\}$ the set of rows and columns. A binary variable $x_{i,j}$ is defined for each square at position (i, j) . $x_{i,j} = 1$ if a queen is placed at that position (i, j) , otherwise $x_{i,j} = 0$.

Let $m = 2n - 3$ be the number of forward/backward diagonals and let $K = \{1, \dots, m\}$ be the set of the diagonals. Constraint (1) ensures that on each row i exactly one queen only can be placed. Constraint (2) ensures that on each column j exactly one queen only can be placed. Constraint (3) makes sure that at most one queen can be placed on every forward diagonal k , and finally, constraint (4) also requires that maximally 1 queen can be placed on every backward diagonal k .

To simplify the notation, we introduce a parameter P_k which is the number of squares on each (forward/backward) diagonal k . It can be defined as:

$$P_k = \min(k + 1, 2n - k - 1), \quad \text{forall } k \in K$$

We also define

$$i' = \max(0, k - n + 1) + j, \quad j' = \max(0, n - k - 1) + j$$

$$i'' = \min(n + 1, k + 2) - j, \quad j'' = \max(0, k - n + 1) + j \quad \text{forall } j \in I, k \in K$$

$$\begin{aligned} & \max \sum_{i,j} x_{i,j} \\ \text{s.t.} \quad & \sum_j x_{i,j} = 1 \quad \text{forall } i \in I \quad (1) \\ & \sum_i x_{i,j} = 1 \quad \text{forall } j \in I \quad (2) \\ & \sum_{j|j \leq P_k} x_{i',j'} \leq 1 \quad \text{forall } k \in K \quad (3) \\ & \sum_{j|j \leq P_k} x_{i'',j''} \leq 1 \quad \text{forall } k \in K \quad (4) \\ & x_{i,j} \in [0, 1] \quad \text{forall } i \in I, j \in I \end{aligned}$$

Note that there are several other implementations in LPL explained in [chess5](#).

Further Comments: The Python code needs the `mip` package. The model is initialized by the function `Model()`. Like in PuLP, the constraints are

“added” to the model using the `+=` operator. Note also the graphical output of LPL.

LPL code (run `queens`)

```

model queens "The n-Queens Problem";
parameter n:=20;
set i,j := 1..n;
binary variable x{i,j}; y; z;
constraint
    Row{i}      : sum{j} x <= 1;
    Col{j}      : sum{i} x <= 1;
    Diag1{k in 1..2*n-3}:
        sum{j|j<=Min(k+1,2*n-k-1)}
            x[Max(0,k-n+1)+j,Max(0,n-k-1)+j] <= 1;
    Diag2{k in 1..2*n-3}:
        sum{j|j<=Min(k+1,2*n-k-1)}
            x[Min(n+1,k+2)-j,Max(0,k-n+1)+j] <= 1;
maximize obj: sum{i,j} x;
Write('The number of Queens is: %3d\n', obj);
Draw.Scale(10,10);
{i,j} Draw.Rect(i,j,1,1,if((i+j)%2,0,1),0);
{i,j|x} Draw.Circle(j+.5,i+.5,.3,5);
end

```

MIP-Python code (download `queens.py`)

```

from sys import stdout
from mip import Model, xsum, BINARY
n = 40
queens = Model()
x = [[queens.add_var('x({},{})'.format(i, j), var_type=BINARY)
      for j in range(n)] for i in range(n)]
# one per row
for i in range(n):
    queens += xsum(x[i][j] for j in range(n)) == 1, 'row
        ({})'.format(i)
# one per column
for j in range(n):
    queens += xsum(x[i][j] for i in range(n)) == 1, 'col
        ({})'.format(j)
# diagonal \
for p, k in enumerate(range(2 - n, n - 2 + 1)):
    queens += xsum(x[i][i - k] for i in range(n)
                    if 0 <= i - k < n) <= 1, 'diag1({})'.
                    format(p)
# diagonal /
for p, k in enumerate(range(3, n + n)):
    queens += xsum(x[i][k - i] for i in range(n)

```

```

        if 0 <= k - i < n) <= 1, 'diag2({})'.
            format(p)

queens.optimize()

if queens.num_solutions:
    stdout.write('\n')
    for i, v in enumerate(queens.vars):
        stdout.write('Q' if v.x >= 0.99 else '.')
        if i % n == n-1:
            stdout.write('\n')

```

5.3.7 JuMP (Julia – open source)

Julia is a high-performance language for scientific computation (see [JuliaLang](#)). While in Python a multitude of packages compete with each other, in Julia exists one dominant package for mathematical modeling: **JuMP**. A somewhat outdated but still convenient book has been written on JuMP with many model examples ([10]).

To install JuMP and solver packages, the package manager Pkg of Julia must be used:

```

using Pkg
Pkg.add("JUMP")
Pkg.add("GLPK")
Pkg.add("HIGHS")
Pkg.add("IPOPT")

```

To install commercial solvers, first install the solver libraries, then add a path and install a package as follows (sometimes the package has to be built). For example, to install the commercial solver CPLEX or Gurobi follow the instructions at [CPLEX package](#) and [Gurobi package](#).

```

ENV["CPLEX_STUDIO_BINARIES"] = "<pathToCplexBin>"
using Pkg
Pkg.add("CPLEX")
Pkg.build("CPLEX")

```

To illustrate modeling in Julia/JuMP, three examples are used.

5.3.7.1 Facility Location (`facilityLoc`)

— Run LPL Code , HTML Document —

Problem: The facility location problem wants to determine whether a facility has to be built and from which facility a customer has to be served at minimal cost. Cost has two components: the transportation costs from a facility to the customer and the building cost (construction cost) of a facility.

Modeling Steps: Let $i \in I$ be a set of customer locations and $j \in J$ a set of facility locations. (x_{ci}, y_{ci}) and (x_{fj}, y_{fj}) are the coordinates of the customers and the facilities. Let f_j be the opening fixed costs for a facility and c_{ij} the cost to transport products from facility j to customer i .

x_{ij} is a binary variable determining whether customer i is assigned to facility j , and y_j is a binary variable indicating whether a facility j is open or not. If a facility j is built ($y_j = 1$) then the building cost is f_j , and if customer i is served from facility j ($x_{ij} = 1$) then the cost is c_{ij} . Hence, minimize the sums over both type of costs. The first constraint says that every customer must be served by exactly one facility and the second constraint makes sure that a customer can only be served by a facility if it is open (if $y_j = 0$ then no customer i can be served from j , that is, x_{ij} must be 0 for all $i \in I$).

$$\begin{aligned} \min \quad & \sum_{i,j} c_{ij} x_{ij} + \sum_j f_j y_j \\ \text{s.t.} \quad & \sum_j x_{ij} = 1 \quad \text{forall } i \in I \\ & x_{ij} \leq y_j \quad \text{forall } i \in I, j \in J \\ & x_{ij} \in [0, 1], y_j \in [0, 1] \quad \text{forall } i \in I, j \in J \end{aligned}$$

Solution: With the random data of the Julia/JuMP code the graphical output of the LPL program is given in Figure 5.6.

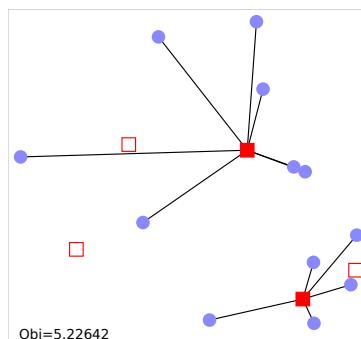


Figure 5.6: Optimal Solution of the Facility Location

Further Comments: The LPL code uses the same random data as the Julia/JuMP code to compare the output graph. The plotting code of Julia was omitted, because it was too long.

The model in Julia is initialized by `Model()` creating an empty model object. In contrast to most Python implementations, JuMP uses the powerful macro-programming of Julia: Variables are declared by a macro `@variable(model, ...)`, constraints are declared by a macro `@constraint(model, ...)`, etc. For the data, Julia's efficient data structure are used. LPL imports the data from the file `facilityLoc.txt` in order to generate the same result as Julia/JuMP.

LPL code (run `facilityLoc`)

```

model facilityLoc "Facility Location";
parameter m; n;
parameter x_c{i}; y_c{i}; x_f{j}; y_f{j};
Read('@facilityLoc.txt', m,n);
set i:=1..m; j:=1..n;
Read{i}('0%1',x_c=2,y_c=3);
Read{j}('0%1',x_f=2,y_f=3);

parameter c{i,j}:=Sqrt((x_c-x_f)^2 + (y_c-y_f)^2);
parameter f{j}:=1;
binary variable y{j}; x{i,j};
constraint C1{i}: sum{j} x = 1;
constraint C2{i,j}: x <= y;
minimize Obj: sum{i,j} c*x + sum{j} f*y;

Draw.Scale(300,-300);
{i,j|x} Draw.Line(x_c,y_c,x_f,y_f);
{i} Draw.Circle(x_c,y_c,.02,5);
{j} Draw.Rect(x_f-.02,y_f-.02,.04,.04,if(y,3,1),3);
Draw.Text('Obj='&Round(Obj,-5),0,0);
end

```

LPL's own random data: Replace the 6 first lines by the this code, if using LPL random data:

```

parameter m:=12; n:=5;
set i:=1..m; j:=1..n;
parameter x_c{i}:=Rnd(0,1); y_c{i}:=Rnd(0,1);
x_f{j}:=Rnd(0,1); y_f{j}:=Rnd(0,1);

```

Julia/JuMP code ([download facilityLoc.jl](#))

```

using JuMP
import HiGHS
import LinearAlgebra
import Plots
import Random

```

```

Random.seed!(314)
m = 12
n = 5
x_c, y_c = rand(m), rand(m)
x_f, y_f = rand(n), rand(n)
f = ones(n);
c = zeros(m, n)
for i in 1:m
    for j in 1:n
        c[i, j] = LinearAlgebra.norm([x_c[i] - x_f[j], y_c[i] - y_f[j]], 2)
    end
end

model = Model(HiGHS.Optimizer)
set_silent(model)
@variable(model, y[1:n], Bin);
@variable(model, x[1:m, 1:n], Bin);
@constraint(model, client_service[i in 1:m], sum(x[i, j]
    for j in 1:n) == 1);
@constraint(model, open_facility[i in 1:m, j in 1:n], x[i
    , j] <= y[j]);
@objective(model, Min, f' * y + sum(c .* x));

optimize!(model)
println("Optimal_value: ", objective_value(model))
# plotting code was omitted

```

5.3.7.2 Urban Planning: A Puzzle ([uplanning](#))

— Run LPL Code , [HTML Document](#) —

Problem: Urban planning is simplified in this model to place either a commercial or a residential lot in a cell of a 5×5 grid. Figure ?? shows an example.

The problem is to reorder the 12 Residential green lots and 13 the Commercial red lots to maximize the quality of the layout. The quality of the layout is determined by a points system. Points are awarded as follows:

- Any column or row that has 5 Residential lots = +5 points
- Any column or row that has 4 Residential lots = +4 points
- Any column or row that has 3 Residential lots = +3 points
- Any column or row that has 5 Commercial lots = -5 points
- Any column or row that has 4 Commercial lots = -4 points



Figure 5.7: An example of Urban Planning

- Any column or row that has 3 Commercial lots = -3 points

The problem was from [Puzzlor](#) – unfortunately, this page has disappeared! So has the Julia/JuMP formulation from the JuMP tutorial page.

Modeling Steps: Let's introduce a set of rows and columns in the grid: $i, j \in I = \{1, \dots, 5\}$. The set of awarded points is: $s \in S = \{1, \dots, 6\}$. Furthermore, we need a set to distinguish rows and columns: $t \in T = \{'r', 'c'\}$. Let's also introduce a data vector a_s for the award points.

The binary variable $x_{i,j}$ is 1 if a cell (i, j) is occupied by a residential lot, otherwise it is occupied by a commercial lot. We need another binary variable $y_{i,s,t}$ which is 1, if and only if the row(column) i has exactly $s - 1$ residential lots. So, for example, $y_{1,1,r} = 1$ means: the row 1 has zero (0) residential lots.

The first constraint says that the number of residential lots is 12:

$$\sum_{i,j} x_{i,j} = 12$$

The second and third constraints are nothing else than *definitions* of the $y_{i,s,t,r}$ variables. The second constraint is for the rows and models the following: if and only if the number of residential lots in row i is $s - 1$ then $y_{i,s,r} = 1$. The third constraint is for the columns and models the following: if and only if the number of residential lots in column i is $s - 1$ then $y_{i,s,r} = 1$. This condition can be formulated as a logical equivalence :

$$y_{i,s,'r'} \leftrightarrow \sum_j x_{i,j} = s - 1 \quad \text{forall } i \in I, s \in S, t \in T$$

$$y_{j,s,'c'} \leftrightarrow \sum_i x_{i,j} = s - 1 \quad \text{forall } j \in I, s \in S, t \in T$$

Finally, the sum of all awards should be maximized:

$$\sum_{i,s,t} a_s \cdot y_{i,s,t}$$

Solution: The total awards that can be reached is 14. Three residential lots are on rows 2 – 5 and columns 1, 2, 4, 5.

obj		14.00			
x{i,j}		1	2	4	5
2		1	1		1
3			1	1	1
4		1	1	1	
5		1		1	1

Further Comments: The Julia/JuMP code is from the JuMP tutorial at [JuMP.dev Tutorial](#)). In Julia/JuMP the problem is formulated explicitly as a linear MIP problem. LPL uses logical constraints that are translated automatically to linear constraints.

The model cannot be found anymore in the JuMP web-site. I guess it is erroneous, although it produces a correct result (but I did not find out its logic!).

LPL code (run [uplanning](#))

```
model uplanning "Urban Planning: A Puzzle";
  set i,j := 1..5 "rows/cols";
  s := 1..6 "point set";
  t := /r,c/ "row/col";
  parameter a{s} := [-5,-4,-3,3,4,5] "awards";
  binary variable x{i,j}; y{i,s,t};
  constraint A: sum{i,j} x = 12;
    R{i,s}: y[i,s,'r'] <-> sum{j} x = s-1;
    C{j,s}: y[j,s,'c'] <-> sum{i} x = s-1;
  maximize obj: sum{i,s,t} a*y;
  Writep(obj,x);
end
```

Julia/JuMP code ([download uplanning.jl](#))

```
using JuMP
import GLPK
import Test
function example_urban_plan()
```

```

model = Model(GLPK.Optimizer)
@variable(model, 0 <= x[1:5, 1:5] <= 1, Int)
rowcol = ["R", "C"]
points = [5, 4, 3, -3, -4, -5]
@variable(model, 0 <= y[rowcol,points,1:5] <= 1, Int)
@objective(
    model,
    Max,
    sum(
        3 * (y["R", 3, i] + y["C", 3, i]) +
        1 * (y["R", 4, i] + y["C", 4, i]) +
        1 * (y["R", 5, i] + y["C", 5, i]) -
        3 * (y["R", -3, i] + y["C", -3, i]) -
        1 * (y["R", -4, i] + y["C", -4, i]) -
        1 * (y["R", -5,i] + y["C", -5,i])) for i in 1:5
    )
)
@constraint(model, sum(x) == 12)
for i in 1:5
    @constraints(model, begin
        y["R",5,i] <= 1 / 5 * sum(x[i,:]) # sum=5
        y["R",4,i] <= 1 / 4 * sum(x[i,:]) # sum=4
        y["R",3,i] <= 1 / 3 * sum(x[i,:]) # sum=3
        y["R",-3,i] >= 1 - 1 / 3 * sum(x[i,:]) # sum=2
        y["R",-4,i] >= 1 - 1 / 2 * sum(x[i,:]) # sum=1
        y["R",-5,i] >= 1 - 1 / 1 * sum(x[i,:]) # sum=0
        y["C",5,i] <= 1 / 5 * sum(x[:,i]) # sum=5
        y["C",4,i] <= 1 / 4 * sum(x[:,i]) # sum=4
        y["C",3,i] <= 1 / 3 * sum(x[:,i]) # sum=3
        y["C",-3,i] >= 1 - 1 / 3 * sum(x[:,i]) # sum=2
        y["C",-4,i] >= 1 - 1 / 2 * sum(x[:,i]) # sum=1
        y["C",-5,i] >= 1 - 1 / 1 * sum(x[:,i]) # sum=0
    end)
)
optimize!(model)
Test.@test termination_status(model) == OPTIMAL
Test.@test primal_status(model) == FEASIBLE_POINT
Test.@test objective_value(model) = 14.0
return
end
example_urban_plan()

```

... continue with Julia code

```

y["C",3,i] <= 1 / 3 * sum(x[:,i]) # sum=3
y["C",-3,i] >= 1 - 1 / 3 * sum(x[:,i]) # sum=2
y["C",-4,i] >= 1 - 1 / 2 * sum(x[:,i]) # sum=1
y["C",-5,i] >= 1 - 1 / 1 * sum(x[:,i]) # sum=0
end)
end
optimize!(model)
Test.@test termination_status(model) == OPTIMAL
Test.@test primal_status(model) == FEASIBLE_POINT
Test.@test objective_value(model) = 14.0
return
end
example_urban_plan()

```

5.3.7.3 The Passport Problem ([passport](#))

— Run LPL Code , HTML Document –

Problem: What is the number of passports a person would need to visit all 199 countries without a visa? (The problem is from [JuMP.dev example](#)).

Modeling Steps: Let $c, d \in C = \{1, \dots, 199\}$ be a set of all 199 countries. And let $p_{c,d}$ be the matrix of passport-index with the following values (jan 2022):

- 3 = visa-free travel
- 2 = eTA is required
- 1 = visa can be obtained on arrival
- 0 = visa is required
- -1 is for all instances where passport and destination are the same

We are looking for entries -1 and 3. So let:

$$P_{c,d} = 1 \quad \text{if } p_{c,d} = -1 \text{ or } p_{c,d} = 3$$

We introduce a binary variable x_c , which is 1 if a passport for this country is needed.

The constraint is as follows: for each country c , at least one valid passport is needed:

$$\sum_d P_{c,d} x_c \geq 1 \quad \text{forall } c \in C$$

The number of passports is to be minimized:

$$\min \sum_c x_c$$

Solution: The result is that one needs 23 passports, with the data $p_{c,d}$ stored in file [passport1.csv](#). They are (Jan 2022) :

Afghanistan	Comoros	Congo	Georgia
Germany	Hong Kong	India	Kenya
Madagascar	Maldives	Mali	New Zealand
North Korea	Papua New Guinea	Singapore	Somalia
Sri Lanka	Tunisia	Turkey	Uganda
United Arab Emirates	United States	Zimbabwe	

Further Comments: Julia uses the package `DataFrames` and `CSV` to read data from Excel. It also uses the package `GLPK` to link the solver. In Julia/JuMP a mathematical model is declared using the macros `@variable`, `@constraint`, and `@objective`.

LPL code (run `passport`)

```
model passport;
set c,d    "countries";
parameter p{c,d} "passport-index"; dummy;
Read('passport1.csv,,\t,',dummy,{c} c);
Read{c}('passport1.csv,,1,\t,',c,{d} p);
// alternatively read from a text file:
//Read{c}('passport.txt,,\t',c=2);
//Read{c}('passport.txt,,\t',c=2,{d} p);
parameter P{c,d}:=if(p=-1 or p=3,1);
binary variable x{c};
constraint A{d}: sum{c} P[c,d]*x[c] >= 1;
minimize nr: sum{c} x;
Write('Min number of passports: %d\n',nr);
Write{c|x}('* %s\n',c);
end
```

Julia/JuMP code (download `passport.jl`)

```
import DataFrames, CSV;
const DATA_DIR = joinpath(\verb?__DIR__, "data")
passport_data = CSV.read(
    joinpath(DATA_DIR, "passport1.csv"),
    DataFrames.DataFrame,
)
function modifier(x)
    if x == -1 || x == 3
        return 1
    else
        return 0
    end
end
for country in passport_data.Passport
    passport_data[!, country] = modifier.(passport_data
    [!, country])
end
C = passport_data.Passport
using JuMP
import GLPK
model = Model(GLPK.Optimizer)
@variable(model, x[C], Bin)
@objective(model, Min, sum(x))
```

```

@constraint(model, [d in C], passport_data[!, d] * x >=
    1)
optimize!(model)
println("Minimum_number_of_passports_needed: ", objective_value(model))
println("Optimal_passports:")
for c in C
    if value(x[c]) > 0.5
        println(" ", c)
    end
end

```

5.3.8 Other Programming Languages

Most solvers also include libraries that can be called from any programming language. The most popular languages are C++, Java, C#, or C. The model formulation is then implemented as a program and the mathematical model structure disappears in the code. The advantage is, of course, the complete flexibility in implementing. If you are a C whiz, you would prefer to use C also to program models. Furthermore, a model may be a small part of a larger software project, then it might be beneficial to use a common language. Many reasons speak for such an approach. Once the code structure of calling the solver is implemented, it is easy to “reuse” it for other projects. On the other hand, most modeling system come along with libraries, so why not include them into the larger project of the chosen programming language? LPL, for instance, exports the whole functionality into a dynamic library callable by most programming languages.

In any case, many reasons guide the choice of a tool for modeling: specifications, habit, skills. In the following, a small linear model is exposed to illustrate the implementation details using Gurobi’s library in the programming language C.

5.3.8.1 A small MIP model ([mip1-c](#))

— Run LPL Code , [HTML Document](#) —

Problem: The problem is the following small linear integer model :

$$\begin{aligned}
 \max \quad & x + y + 2z \\
 \text{s.t} \quad & x + 2y + 3z \leq 4 \\
 & x + y \geq 1 \\
 & x, y, z \in [0, 1]
 \end{aligned}$$

The optimal solution is $(x, y, z) = (1, 0, 1)$. The example is from [Gurobi Examples](#).

Further Comments: LPL calls Gurobi's library to solve the model. Internally, it is basically the same code as the C code below, but all those implementation details are hidden from the modeler. The C code first declares the data needed. Then the model is initialized, license is checked, some parameters are set using the library functions GRBemptyenv, GRBsetstrparam, GRBstartenv, GRBnewmodel. After the initialization, variables and constraints are added with GRBaddvars and GRBaddconstr, the optimizer is called (GRBoptimize) and finally various output functions extract the solution.

That is basically the same as LPL does when calling the Gurobi solver.

LPL code (run [mip1-c](#))

```
model mip1c "A small MIP model";
binary variable x; y; z;
maximize O: x + y + 2*z;
constraint C1: x + 2*y + 3*z <= 4;
              C2: x + y           >= 1;
Writep(O,x,y,z);
end
```

C code for gurobi ([download mip1-c.c](#))

```
// Copyright 2022, Gurobi Optimization, LLC
#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int main(int argc, char *argv[]) {
    GRBEnv *env = NULL;
    GRBmodel *model = NULL;
    int error = 0;
    double sol[3];
    int ind[3];
    double val[3];
    double obj[3];
    char vtype[3];
    int optimstatus;
    double objval;

    error = GRBemptyenv(&env); if (error) goto QUIT;
    error = GRBsetstrparam(env, "LogFile", "mip1.log"); if
(error) goto QUIT;
    error = GRBstartenv(env); if (error) goto QUIT;
    error = GRBnewmodel(env, &model, "mip1", 0, NULL, NULL,
NULL, NULL, NULL); if (error) goto QUIT;
```

```
// Add variables
obj[0] = 1; obj[1] = 1; obj[2] = 2;
vtype[0] = GRB_BINARY; vtype[1] = GRB_BINARY; vtype[2]
    = GRB_BINARY;
error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, obj,
    NULL, NULL, vtype, NULL);
if (error) goto QUIT;

// Change objective sense to maximization
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE,
    GRB_MAXIMIZE);
if (error) goto QUIT;
```

C code for gurobi, continuing ...

```
// First constraint: x + 2 y + 3 z <= 4
ind[0] = 0; ind[1] = 1; ind[2] = 2;
val[0] = 1; val[1] = 2; val[2] = 3;

error = GRBaddconstr(model, 3, ind, val, GRB_LESS_EQUAL
    , 4.0, "c0");
if (error) goto QUIT;

// Second constraint: x + y >= 1
ind[0] = 0; ind[1] = 1;
val[0] = 1; val[1] = 1;

error = GRBaddconstr(model, 2, ind, val,
    GRB_GREATER_EQUAL, 1.0, "c1");
if (error) goto QUIT;

// Optimize model
error = GRBoptimize(model);
if (error) goto QUIT;

// Write model to 'mip1.lp'
error = GRBwrite(model, "mip1.lp");
if (error) goto QUIT;

// Capture solution information
error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &
    optimstatus);
if (error) goto QUIT;

error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &
    objval);
if (error) goto QUIT;

error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3,
    sol);
```

```

if (error) goto QUIT;

printf("\nOptimization_complete\n");
if (optimstatus == GRB_OPTIMAL) {
    printf("Optimal_objective:_%.4e\n", objval);
    printf("_x=% .0f,_y=% .0f,_z=% .0f\n", sol[0], sol[1],
           sol[2]);
} else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model_is_infeasible_or_unbounded\n");
} else {
    printf("Optimization_was_stopped_early\n");
}

QUIT:
if (error) {
    printf("ERROR:_%s\n", GRBgeterrormsg(env)); exit(1);
}
// Free model
GRBfreemodel(model);
// Free environment
GRBfreeenv(env);
return 0;
}

```

5.4. Further Tools

The reader finds a lot of other mathematical tools and toolboxes in the Internet. I have basically concentrated on numeric optimization. Further tools I found interesting are given below.

- **SageMath** is an open source mathematical software built out of nearly 100 open-source packages to study elementary and advanced, pure and applied mathematics. This includes a huge range of mathematics, including basic algebra, calculus, elementary to very advanced number theory, cryptography, numerical computation, commutative algebra, group theory, combinatorics, graph theory, exact linear algebra and much more. SageMath can also be linked to most advanced optimization libraries like Gurobi or Cplex.
- **Scilab** is another open source software for numerical computation. It is -like **GNU Octave** an open-source alternative to MATLAB. It also contains a high-level, numerically oriented programming language.
- A very popular open-source environment for developing models is **Jupyter Notebook**. All mentioned systems in Python or Julia can be neatly be integrated with Jupyter Notebook. But other languages can be used: Go, Scala, Erlang, etc. For teaching math modeling this is an excellent

tool. If you like to model with JuMP/Julia, for instance, you only need to install the IJulia package:

```
using Pkg  
Pkg.add("IJulia")  
using IJulia  
notebook()
```

Now you are ready to use the Notebook to interactively enter the model code.

- **The R-Project** is a open-source tool containing a programming language basically for statistical applications. But it has also interface to optimization tools.
- The **NEOS Server** is an Internet-based client-server application that provides free access to more than 60 libraries of optimization solvers. A model can be submitted in various formats (GAMS, AMPL, and others).
- **Mathcad Prime** was the first to introduce 1986 live editing (WYSIWYG) of typeset mathematical notation in an interactive notebook, combined with automatic computations. A idea copied by many other systems. The current commercial version contains many features in numerical and symbolic mathematics, visualization, etc. It is not as powerful as Maple or Mathematica.
- Besides of the already mentioned solver libraries (Gurobi, Cplex, XPress), **HiGHS** is one of the most advanced open-source linear optimization solver. **MOSEK** is a commercial solver library and solves LPs, QPs, SOCPs, SDPs and MIPs. It includes interfaces to C, C++, Java, MATLAB, .NET, Python and R. **Knitro** is one of the most advanced solver for nonlinear optimization.
- If you just need a powerful mathematical library containing high performance algorithms and you program everything else yourself, consider the **Nag Library**.

5.5. Conclusion

The list of mathematical tools in this paper is uncomplete and it is based on my personal and subjective experiences. I hope that I mentioned the most important once. Furthermore, there is no rating here, just a simple example is given that should give a first glimpse on the tools.

CHAPTER 6

VARIOUS MODEL TYPES

“The only free cheese is in the mousetrap.”

Mathematical models can be classified into groups using various criteria: discrete, continuous, linear, non-linear, etc. Different model types also dominate in different research domains. Linear and non-linear optimization models prevail the realm of operations research. Dynamical models such as differential equations, appear more in physics and engineering. Statistical models and models treating uncertainty (p.e. stochastic models, fuzzy models) occur in social sciences. This is not to say that physicists do not use statistics. However, the various research domain use a slightly different vocabulary in building their models.

Coming from the operations research, a lot of attention is given to the classification in this research field. Several model types are formulated in mathematical notation and in the modeling system LPL (see [72]). They are compared with each other from various point of views. Concrete model application examples are given for most model type.

A ZIP file of all models can be found [HERE](#)

6.1. Introduction

Models can be classified into different groups using various criteria:

Qualitative versus quantitative: When taking a decision or solving a problem, many trust one's gut, taking into account intuition and experiences. We all build consciously or unconsciously a model to capture the situation at hand. More often than not, the model may have the form of a feeling and a decision is taken spontaneously. On the other hand, collect all kinds of relevant data, formulate the conditions in a clear way, define the goals formally may lead to superior decision. All depends on the situation and the problem. Surely, many problems must be quantified to achieve a "good" result. For example, to planify a round-trip to deliver goods to various clients, intuition will in general be a bad advisor in finding the shortest trip. This paper only treats quantitative models.

Continuous versus discrete (integer or Boolean): Fractional values may be unacceptable. "Buy 3.5 aeroplanes" does not make sense in most context, but "on average 3.5 persons die every second due to drug consumption" may be adequate. However, there is another huge and important class of models that use discrete (Boolean) values: models to represent problems that must answer yes/no questions, for example, "should we build a factory or not?". The significance of this class is shown in another paper (see [68]). Discrete – also called combinatorial – problems are in general much more difficult to solve. On the other hand, problems dealing with physical quantities and their rates of change can be modeled as differential equations. Most of them can only be solved by discretizing them.

Zero- versus One- versus Multi-objective: We may looking for *all*, for an *arbitrary* or for a *particular* solution. For zero-objective problems, we are interested only in an arbitrary solution, it may be unique or not. For example, in model [example3a](#) any solution would be fine (it is unique in this case). In many situations, we are interested in the “best” solution, with regards to one or many objectives. These problems are called *optimization problems*. We may minimize cost, resources, redundancy, etc. or we may maximize revenue, utility, turnover, customer satisfaction, robustness, etc. Normally, we are looking for one objective. In same cases, however, several – maybe conflicting – objectives have to be considered. These problems are called multi-objective optimization problems. Such problems can be handled by various methods, for example, with *goal programming* (see below).

Unconstraint versus constrained: Within the class of optimization problems there are unconstraint or constraint problems. Unconstraint problems only contain an objective function and no constraint, example: find the minimum of a parable : $\min f(x) = x^2$. Constraint problems consist of an objective and one or several constraints.

Deterministic versus stochastic: If the data in a resultant model is not known with certainty, we have stochastic models. In many situation, the data are uncertain or unknown, for example, future demand on a market are not known or uncertain, but even data from the past are often not known or only a small sample is known. Statistical methods to estimate them are then needed. Mostly, however, we pretend that data are deterministic in order to avoid complicated models. In many cases, this may be realistic, but the modeler should be aware of that fact. There is a broad theory of stochastic models, but in this paper only deterministic once are treated.

Static versus dynamical: In many situation we are looking for an (optimal) state: finding an optimal production plan, a tournament schedule of a sport league, the shortest round trip, etc. Normally, optimization models result from these problems. These models are static. When change has to be modeled, like motion over time in physics, evolution in animal population in biology, fluctuation in monetary quantities in economy, or development of a virus in a pandemic, dynamical models are commonly use, such as discrete dynamical systems or a system of differential equations.

Linear versus non-linear: Linear models only contain linear terms with regards to the variables. Completely different methods are used to solve linear and non-linear problems. Linear, continuous problems are solved mainly by the simplex method, a modification of this method solves also certain problems containing quadratic terms (QP, QPC, etc.) – if they are convex. Linear, discrete (combinatorial) problems use branch-and-bound, cutting plane or other reduction algorithms. Non-linear problems are in general much more difficult to solve, and a large number of algorithms have been developed. The distinction between linear and non-linear models may be arbitrary. We also may partition the models into *convex and concave*, or into “*easy*” and “*difficult*”

to solve. “Easy” could be defined as problems in P (polynomial-solvable), the “difficult” once are NP-complete, or beyond.¹ In any case, the purpose of these partitions is to classify the models into groups of different algorithmic methods. The reason is more practical than theoretical: A modeling system – that allows to *formulate* a wide range of mathematical models, must be able to be linked to a large number of solutions algorithms – so called *solvers* – in order to solve it. The modeling system should be able to recognize into which group(s) a model falls to select the solver automatically.

In this paper, a general overview of various mathematical (optimization) model types is presented. A general specification and formulation is given first in a mathematical notation then in the modeling language LPL (see [72]).

A mathematical model has the following general form:

$$\begin{array}{ll} \min & f(x) \\ \text{subject to} & g_i(x) \leq 0 \quad \text{forall } i \in \{1, \dots, m\} \\ & x \in X \end{array}$$

If the first line is missing, we have a zero-objective model, if the second line is missing, we have an unconstraint model. The f and g_i are functions defined in \mathbb{R}^n .² X is a subset of \mathbb{R}^n (or \mathbb{N}^n), and x is a vector of n components x_1, \dots, x_n . The above problem has to be solved for the values of the *variables* x_1, \dots, x_n that satisfy the restrictions g_i while minimizing the function f . The function f is called *the objective function*, g_i are *the constraints*. A vector $x \in X$ that satisfies all constraints $g_i(x) \leq 0$ is called *feasible solution*. The collection of all such points is *the feasible region*. The problem then of the mathematical model above is to find a x^o such that $f(x) \geq f(x^o)$ for each feasible point x . Such a point x^o is called *an optimal solution*.

A small example is the following model (see [4], page 3):

$$\begin{array}{ll} \min & (x_1 - 3)^2 + (x_2 - 2)^2 \\ \text{subject to} & x_1^2 x_2 - 3 \leq 0 \\ & x_2 - 1 \leq 0 \\ & -x_1 \leq 0 \end{array}$$

The objective function and the 3 constraints are:

¹ It would be too involving here to explain these concepts. They are developed in the theory of complexity in computer science.

² The set of operators that can be used in a function decide on the expressiveness on models. Boolean operators belong to that set, if all kind of combinatorial problems are to be formulated. Note that any model containing several constraints could be expressed by a single constraint – by concatenating the constraints with the Boolean operator **and**.

$$\begin{aligned}
 f(x_1, x_2) &\text{ is } (x_1 - 3)^2 + (x_2 - 2)^2 \\
 g_1(x_1, x_2) &\text{ is } x_1^2 x_2 - 3 \leq 0 \\
 g_2(x_1, x_2) &\text{ is } x_2 - 1 \leq 0 \\
 g_3(x_1, x_2) &\text{ is } -x_1 \leq 0
 \end{aligned}$$

Figure 6.1 illustrates the model geometrically in the two-dimensional *real Euclidean space*.

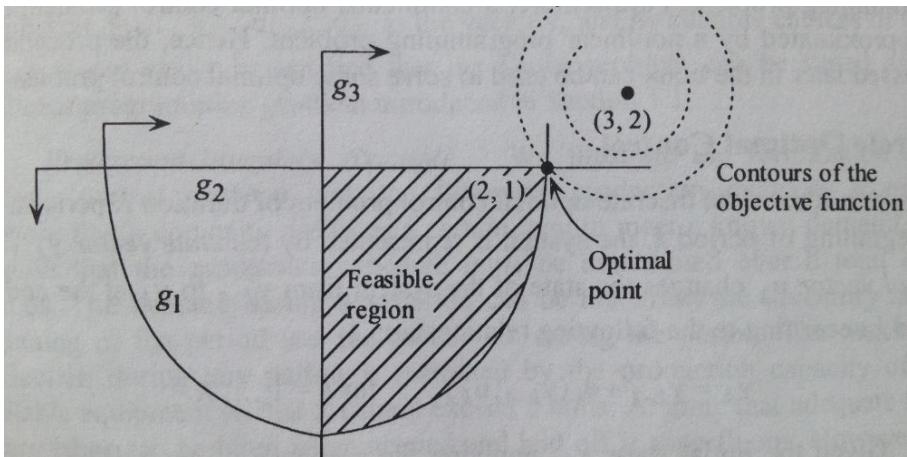


Figure 6.1: Geometric representation of the model

6.2. Variations

There are useful practical variations in notation that can be reduced to a standard model type.

1. A maximizing objective function can be transformed into a minimizing function (and vice versa):

$$\max f(\mathbf{x}) \implies \min -f(\mathbf{x})$$

A greater or equal constraint can be transformed into a less or equal than constraint:

$$f(\mathbf{x}) \geq 0 \implies -f(\mathbf{x}) \leq 0$$

A equality constraint can be transformed into a less and a greater or equal than constraint:

$$f(\mathbf{x}) = 0 \implies \begin{cases} f(\mathbf{x}) \geq 0 \\ f(\mathbf{x}) \leq 0 \end{cases}$$

2. A maximin (or a minimax) objective can be transformed as follows (let $I = \{1, \dots, n\}$) (note that the functions are convex):

$$\max \left(\min_{i \in I} f(x_i) \right) \Rightarrow \begin{cases} \max & z \\ \text{s.t.} & f(x_i) \leq z \quad \text{forall } i \in I \\ & z \geq 0 \end{cases}$$

$$\min \left(\max_{i \in I} f(x_i) \right) \Rightarrow \begin{cases} \min & z \\ \text{s.t.} & f(x_i) \geq z \quad \text{forall } i \in I \\ & z \geq 0 \end{cases}$$

This notation is useful for example in zero-sum games (see [gameh](#), or in regression models and others (see model [regression](#))).

3. The following objective function is concave, hence, the transformation is more involving (M being an upper bound for x):

$$\max |x| \Rightarrow \begin{cases} \max & z \\ \text{s.t.} & z = x_p + x_n \\ & x = x_p - x_n \\ & x_p = My \\ & x_n = M(1-y) \\ & y \in \{0, 1\} \\ & z, x_p, x_n \geq 0 \end{cases}$$

4. An arbitrary (non-linear) objective function $f(x)$ can be replaced by a linear function by introducing an additional variable z . Then the general model is replaced by:

$$\begin{array}{ll} \min & z \\ \text{subject to} & g_i(x) \leq 0 \quad \text{forall } i = 1, \dots, m \\ & x \in X \\ & f(x) \leq z \end{array}$$

5. A fractional linear objective can be replaced by a linear one by introducing additional variables. The method is explained in model [bill046](#)
6. Let $f(y)$ be a non-linear function used in a constraint. Then we can approximate it by a piecewise linear function. The procedure is explained in model [bill227](#).
7. The variables x in a standard (non-linear) model are free. If they must be positive only, then one can easily add the constraints: $(x \geq 0)$.

The variables in a linear model (LP) normally are tacitly limited to be positive. If a variable x should be free, then we can make the following substitutions:

$$x = x_p + x_n, \quad \text{with} \quad x_p, x_n \geq 0$$

8. A model with k *multiple objectives*, say $f_1(x), \dots, f_k(x)$ can be formulated by forming a new combined objective f as follows:

$$f(x) = w_1 f_1(x) + \dots + w_k f_k(x)$$

where w_1, \dots, w_k are numbers (weights) that reflect the importance of the single objectives (the higher the number, the more important the objective). An illustrative example is give in the model [multi1](#).

9. A useful variant is **goal programming** by the means of *soft constraints*. In many situation, we do not mind if a particular constraint is slightly violated. For instance, with a given budget of \$1'000'000, it would certainly be acceptable to over- or undershoot it by, say, \$100. On the other hand, a constraint that defines Expenses = Budget, enforces *exactly* the amount. To resolve (and relax) this problem, the constraint can be made "soft", that is, two positive variables p and n are added to absorb a positive or negative deviation (slacks) from the budget goal, and the constraint is modified to

$$\text{Expenses} + n - p = \text{Budget} \quad (n, p \geq 0)$$

In the objective function, a weighted sum of these slack variables can be minimized (as seen in the previous item about multiple objectives, or the maximal deviation can be minimized. Hence, the goal, namely "attaining the budget exactly" is substituted by the goal "attaining the budget approximately". Several positive and negative slacks could be added to a goal constraint, that can be penalized with an increasing weight parameters in the objective function. This method is closely related with the problem of multiple objectives: The objectives are – between others – to minimize the deviations of the different goals. A example to illustrate this technique is given in the models [goalpr](#) and in [goalpr1](#).

10. *Preemptive goal programming* is another technique to handle soft constraints. Finding the right weights for the different goals in an multiple objective function is sometimes difficult. Hence, in this method the user orders the goals in a list of decreasing importance. The first optimization minimizes/maximizes the most important goal. Then the corresponding slack variables are fixed and the next goal is optimized, and so on, until all goals has been optimized. Formally, solve the model first with $f_1(x)$ (supposing f_1 is the most important goal). Let the optimal value be f_1^0 , then add the constraint $g_{m+1}(x) = f_1(x) - f_1^0 = 0$. Now continue in the same way with the second most important objective, an so on until the least important objective. This method is illustrated by the model [goalpr2](#). A real model where this method is used can be found in the [Casebook II](#).

In an application, the **objective function** may have various meaning. We may *maximize* profit, utility, turnover, return on investment, net present value,

number of employees, customer satisfaction, probability of survival, robustness; or we may *minimize* cost, number of employees, redundancy, deviations, use of resources, etc.

The **constraints** reflect a large variation of requirements in a concrete application. In a production context there may be *capacity*, *material availability*, *marketing limitation*, *material balance* constraints; in a resource scheduling we may have *due date*, *job sequencing*, *space limitation* constraints, etc.

In the following sections various model types are presented and most of them implemented:

1. *Linear programs (LP)* consist of linear constraints and a linear objective function and real variables.
2. *Integer (linear) programs (IP)* consist of linear constraints and a linear objective function and integer variables.
3. *0-1 (linear) programs* consist of linear constraints and a linear objective function and binary variables (integer variables with only values of 0 and 1). This type is a special case of the previous one, where variables are integer but can only take the values 0 or 1. From the formulation point of view, the difference between the three types is very small. However – as we shall see – the difficulty to solve integer problems is much higher. Linear programs (LP) can be solved in polynomial time, while IP and 0-1 IP problems are mostly NP-complete. The application field of these three model types is very large. We shall point to examples.
4. *Quadratic problems (QP)* which consist of linear constraints and a quadratic convex objective function.
5. *0-1 quadratic problems (0-1-QP)* which consist of linear constraints and a quadratic convex objective function and contain 0-1 variables. Both types have interesting applications in portfolio theory.
6. *Quadratic constraint problems (QCP)* which consist of linear and quadratic convex constraints and a quadratic convex objective function.
7. *Second order cone problems (SOCP)*, which have many applications in physics. All of these previous model classes are convex problems – and much easier to solve.
8. *Non-convex quadratic problems (NCQP)*, an interesting application is given below. All of these previous model classes can today be solved by commercial solvers like **Gurobi** or **Cplex**.
9. A large class of models is the *non-linear optimization model class (NLP)*. Many solver exist for specific subclasses of this class. Also some general

commercial solvers exist such as Knitro or Hexaly solver. Version 11.0 of Gurobi also can solve some non-linear models.

10. Many application in physics, biology, or economy use dynamic models which are basically discrete *dynamic system* or systems of differential equations.
11. A particular class is the *permutation model class*. Many routing, scheduling, or assignment problems can be formulated in this way, a concrete example is given, more of them can be found in the paper [71].

For most of the model types, a concrete application and an implementation in LPL is given. We shall show the gap in difficulty to solve integer problems compared to non-integer linear problems.

Of course, one can also build combined models: LP model part mixed with IP model leads to *mixed integer programs (MIP)* containing real *and* integer variables, etc.

6.3. A Linear Program (`examp-lp`)

— Run LPL Code , HTML Document –

Problem: The general **linear programming** model – called **LP** – consists of a linear objective function $f(x)$ and m linear constraints $g_i(x)$ and n variables. It can be compactly formulated as follows (see [83]):

$$\begin{aligned} \min \quad & \sum_{j \in J} c_j x_j \\ \text{subject to} \quad & \sum_{j \in J} a_{i,j} x_j \geq b_i && \text{forall } i \in I \\ & x_j \geq 0 && \text{forall } j \in J \\ & I = \{1 \dots m\}, J = \{1 \dots n\}, m, n \geq 0 \end{aligned}$$

An more compact formulation using matrix notation for the model is as follows:

$$\begin{aligned} \min \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} \quad & \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

The objective function f the constraints g_i , and \mathbf{X} (in the general format) are as follows:

$$\begin{aligned} f(x_1, \dots, x_n) &= c_1 x_1 + \dots + c_n x_n \\ g_i(x_1, \dots, x_n) &= b_i - (a_{i,1} x_1 + \dots + a_{i,n} x_n) \leq 0 && \text{forall } i = 1, \dots, m \\ \mathbf{x} &\in \mathbf{R}^n \end{aligned}$$

Note that in addition to the g_i constraints we also have the non-negativity conditions on the variables x in the standard formulation. If we need negative variables, we must explicitly replace x by $x_1 - x_2$, where $x_1, x_2 \geq 0$ are two positive vectors $\in \mathbf{R}^n$.

In the following, a concrete problem with random data for the matrix \mathbf{A} , and the two vectors \mathbf{b} and \mathbf{c} is specified (with $n = 15$ and $m = 15$):

$$\mathbf{c} = (16 \ 73 \ 39 \ 81 \ 68 \ 60 \ 90 \ 11 \ 33 \ 89 \ 71 \ 12 \ 24 \ 23 \ 47)$$

$$\mathbf{A} = \begin{pmatrix} 87 & 34 & 0 & 0 & 43 & 0 & 52 & 85 & 36 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 39 & 0 & 84 & 0 & 0 & 88 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\ 55 & 63 & 79 & 0 & 0 & 0 & 0 & 71 & 0 & 0 & 0 & 0 & 70 & 0 & 79 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 73 & 0 & 0 & 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 66 & 0 & 0 & 34 & 0 & 0 & 24 & 0 & 61 & 0 & 0 & 64 \\ 0 & 0 & 16 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 15 & 70 & 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 52 & 0 & 14 & 0 & 0 & 92 & 0 & 58 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 98 & 0 & 19 & 0 & 44 \\ 0 & 70 & 37 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 86 & 0 & 93 & 0 \\ 90 & 0 & 82 & 0 & 66 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 \\ 77 & 30 & 0 & 0 & 0 & 0 & 13 & 10 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 72 & 0 & 0 & 0 & 79 & 0 & 0 & 0 & 0 & 0 & 11 & 90 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 & 55 & 0 & 23 & 0 & 0 & 0 & 0 & 0 & 36 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 119 \end{pmatrix}$$

With these data, the model is specified by the following explicit linear program:

$$\begin{array}{ll} \text{min} & 16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 + 33x_9 \\ & + 89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\ \text{subject to} & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\ & 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\ & 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\ & 73x_8 + 22x_{12} \geq 0 \\ & 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\ & 16x_3 + 19x_4 + 18x_{15} \geq 0 \\ & 15x_1 + 70x_2 + 61x_3 \geq 0 \\ & 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\ & 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\ & 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\ & 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\ & 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\ & 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\ & 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\ & x_j \geq 0 \quad \text{forall } j \in \{1 \dots 15\} \end{array}$$

Modeling Steps: The formulation of the model in the LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First, define the two sets i and j . Then declare and assign the data as parameters A , c , and b . The variable vector x is declared, and finally the constraints R and the minimizing objective function ob are written.

Note that the data matrices **A**, **b**, and **c** are generated using LPL's own random generator. (To generate each time the same data, the code can also use the function `SetRandomSeed(a)` where `a` is an integer.)

Listing 6.1: The Complete Model implemented in LPL [72]

```
model Lp15 "A Linear Program";
  set i := [1..15];  j := [1..15];
  parameter A{i,j}:= Trunc(if(Rnd(0,1)<.25,Rnd(10,100)));
    c{j} := Trunc(Rnd(10,100));
    b{i} := Trunc(if(Rnd(0,1)<.15,Rnd(0,120)));
  variable x{j};
  constraint R{i} : sum{j} A*x >= b;
  minimize obj : sum{j} c*x;
  Writep(obj,x);
end
```

Today, models with $n, m > 10000$ and much larger are solved on a regular base. LP models with millions of variables can be solved today. A linear programming model with 2000 variables and 1000 linear constraints can be downloaded and solved at: [lp2000](#).

Solution: The small model defined above has the following solution:

$$x = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1.4667 \ 1.1154 \ 0 \ 0 \ 0 \ 0 \ 0 \ 2.5929)$$

The optimal value of the objective function is:

$$\text{obj} = 174.8096$$

Further Comments: The linear programming (LP) model has many applications in quantitative decision making. LP is used for capacity planning, resource (raw material) allocation, manpower planning, blending, transportation, network flow, network design, portfolio selection, optimal marketing mix, multiperiod product mix, and many others.

A small example in multi period production planning is given in the model: [product](#). A historical example of a the so-called min-cut problem is modeled by [Tolstoi1930](#). An application example for transportation is [trans](#) or [ship3](#).

A linear model implementing three common regression methods is given in [regression](#).

Further Notes: As already seen in the model *regression* certain non-linear functions can be transformed in a way that the model becomes an linear one. We mention three of them (see [9], Chap. 13):

1. The **absolute value**. Suppose the (minimizing) objective function has the form:

$$f(\mathbf{x}) = |\mathbf{c} \cdot \mathbf{x}|$$

One can add a single variable y and modify the LP as follows:

$$\begin{array}{lll} \min & y \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} & \geq \mathbf{b} \\ & \mathbf{c} \cdot \mathbf{x} & \leq y \\ & -\mathbf{c} \cdot \mathbf{x} & \leq y \\ & \mathbf{x} & \geq 0 \\ & y & \geq 0 \end{array}$$

This can be generalized. If the objective function is:

$$f(\mathbf{x}) = \sum_{i \in I} |\mathbf{c}_i \cdot \mathbf{x}|$$

One can add a vector of variables y_i (with $i \in I$) and modify the LP as follows:

$$\begin{array}{lll} \min & \sum_{i \in I} y_i \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} & \geq \mathbf{b} \\ & \mathbf{c}_i \cdot \mathbf{x} & \leq y_i, \text{ for all } i \in I \\ & -\mathbf{c}_i \cdot \mathbf{x} & \leq y_i, \text{ for all } i \in I \\ & \mathbf{x} & \geq 0 \\ & y_i & \geq 0, \text{ for all } i \in I \end{array}$$

2. The **maximal value**. Suppose the objective function has the form:

$$f(\mathbf{x}) = \max \mathbf{c} \cdot \mathbf{x}$$

One can add a single variable y and modify the LP as follows:

$$\begin{array}{lll} \min & y \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} & \geq \mathbf{b} \\ & \mathbf{c} \cdot \mathbf{x} & \leq y \\ & \mathbf{x} & \geq 0 \\ & y & \geq 0 \end{array}$$

In a similar way, if the objective function is:

$$f(\mathbf{x}) = \max_{i \in I} |\mathbf{c}_i \cdot \mathbf{x}|$$

One can add a single variable y and modify the LP as follows:

$$\begin{array}{lll} \min & \sum_{i \in I} y_i \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} & \geq \mathbf{b} \\ & \mathbf{c}_i \cdot \mathbf{x} & \leq y, \text{ for all } i \in I \\ & -\mathbf{c}_i \cdot \mathbf{x} & \leq y, \text{ for all } i \in I \\ & \mathbf{x} & \geq 0 \\ & y_i & \geq 0, \text{ for all } i \in I \end{array}$$

3. The **fractional LP**. Suppose the objective function consists of maximizing the ratio of two linear functions (where \mathbf{p} and \mathbf{q} are two data vectors and α and β are two scalars):

$$f(\mathbf{x}) = \frac{\mathbf{p} \cdot \mathbf{x} + \alpha}{\mathbf{q} \cdot \mathbf{x} + \beta}$$

One can convert this into a LP model. If the feasible set $\{\mathbf{x} | \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}, \mathbf{x} \leq 0\}$ is nonempty and bounded and if $\mathbf{q} \cdot \mathbf{x} + \beta > 0$, using the following transformations:

$$z = \frac{1}{\mathbf{q} \cdot \mathbf{x} + \beta} \quad , \quad \mathbf{y} = z\mathbf{x}$$

we obtain the following LP:

$$\begin{array}{lll} \min & \mathbf{p} \cdot \mathbf{x} + \alpha \\ \text{subject to} & \mathbf{A} \cdot \mathbf{y} - \mathbf{b}z & \leq 0 \\ & \mathbf{q} \cdot \mathbf{y} + \beta z & = 1 \\ & \mathbf{y} & \geq 0 \\ & z & \geq 0 \end{array}$$

A small example is given in [bill046](#).

6.4. A Integer Linear Program (examp-ip)

— Run LPL Code , HTML Document –

Problem: The general **linear integer programming** model – called **IP** – contains a linear objective function $f(x)$, m linear constraints $g_i(x)$, and n integer variables. It can be compactly formulated as follows (see [83]):

$$\begin{array}{ll} \min & \sum_{j \in J} c_j x_j \\ \text{subject to} & \sum_{j \in J} a_{i,j} x_j \geq b_i \quad \text{forall } i \in I \\ & x_j \in \mathbb{N}^+ \quad \text{forall } j \in J \\ & I = \{1 \dots m\}, J = \{1 \dots n\}, m, n \geq 0 \end{array}$$

A more compact formulation using matrix notation for the model is :

$$\begin{array}{ll} \min & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \in \mathbb{N}^n \end{array}$$

The objective function f , the constraints g_i , and x (in the general model format) are as follows:

$$\begin{aligned} f(x_1, \dots, x_n) &= c_1 x_1 + \dots + c_n x_n \\ g_i(x_1, \dots, x_n) &= b_i - (a_{i,1} x_1 + \dots + a_{i,n} x_n) \leq 0 \quad \text{forall } i = 1, \dots, m \\ x &\in \mathbb{N}^n \end{aligned}$$

The IP model has the same notation as the LP model, the only difference is that the variables are integer values. However, IP model are much more difficult to solve in general. (To get an idea solve the following model: [lp2000](#). Then try to solve [ip2000](#).)

In the following a problem with random data is specified (with $n = 15$ and $m = 15$):

$$c = (16 \ 73 \ 39 \ 81 \ 68 \ 60 \ 90 \ 11 \ 33 \ 89 \ 71 \ 12 \ 24 \ 23 \ 47)$$

$$A = \begin{pmatrix} 87 & 34 & 0 & 0 & 43 & 0 & 52 & 85 & 36 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 39 & 0 & 84 & 0 & 0 & 88 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\ 55 & 63 & 79 & 0 & 0 & 0 & 0 & 71 & 0 & 0 & 0 & 0 & 70 & 0 & 79 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 73 & 0 & 0 & 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 66 & 0 & 0 & 34 & 0 & 0 & 24 & 0 & 61 & 0 & 0 & 64 \\ 0 & 0 & 16 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 15 & 70 & 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 52 & 0 & 14 & 0 & 0 & 92 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 98 & 0 & 19 & 0 & 0 \\ 0 & 70 & 37 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 86 & 0 & 93 & 0 \\ 90 & 0 & 82 & 0 & 66 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 \\ 77 & 30 & 0 & 0 & 0 & 0 & 13 & 10 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 72 & 0 & 0 & 0 & 79 & 0 & 0 & 0 & 0 & 0 & 11 & 90 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 & 55 & 0 & 23 & 0 & 0 & 0 & 0 & 0 & 36 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 58 \\ 44 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 119 \end{pmatrix}$$

The data given above specify the following explicit linear program:

$$\begin{aligned} \text{min} \quad & 16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 + 33x_9 \\ & + 89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\ \text{subject to} \quad & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\ & 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\ & 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\ & 73x_8 + 22x_{12} \geq 0 \\ & 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\ & 16x_3 + 19x_4 + 18x_{15} \geq 0 \\ & 15x_1 + 70x_2 + 61x_3 \geq 0 \\ & 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\ & 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\ & 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\ & 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\ & 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\ & 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\ & 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\ & x_j \in \mathbf{N}^+ \quad \text{forall } j \in \{1 \dots 15\} \end{aligned}$$

Modeling Steps: The formulation of the model in LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First we define the two sets i and j . Then we declare and assign the data as parameters A , c , and b . The variable vector x is declared with the keyword `integer` (the only difference to the LP model), and finally the constraints R and the minimizing objective function `obj` are written.

Note that the data matrices **A**, **b**, and **c** are generated using LPL's own random generator. (To generate the same data each time, the code can also use the function `SetRandomSeed(a)` where `a` is an integer.)

Listing 6.2: The Complete Model implemented in LPL [72]

```

model Ip15 "A Integer Linear Program";
  set i := [1..15]; j := [1..15];
  parameter A{i,j} := Trunc(if(Rnd(0,1)<0.25, Rnd(10,100)
    ));
    c{j} := Trunc(Rnd(10,100));
    b{i} := Trunc(if(Rnd(0,1)<0.15, Rnd(0,120))
      );
  integer variable x{j};
  constraint R{i} : sum{j} A*x >= b;
  minimize obj : sum{j} c*x;
  Writep(obj,x);
end

```

Solution: The small model defined above has the following solution :

$$x = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 2 \ 4 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)$$

The optimal value of the objective function is:

$$\text{obj} = 201$$

If we compare this solution with the corresponding LP model in [examp-lp](#), we notice that the objective value is much larger (201 compared to 174.8096). Furthermore, the vector x is not simply the “round-down” of the LP solution, as one may expect. To understand why we cannot simply round up or down to get an integer solution from a corresponding LP model, open the model [willi155](#) and read the modeling text to understand why.

Further Comments: IP problems are *much* more difficult to solve than LP problems in general. However, there is a surprisingly rich application field for integer programs. Obvious applications are problems where we have indivisible objects, such as number of persons, machines, etc. However, these are not typical applications. If we have the number of drivers in a small company then we may model this as an integer quantity, however if we have the population in a country then we may approximate it as a continuous variable. This obvious aspect, however, does *not* reveal the real power of integer programming. We really need integer programming in the four following contexts in this order of importance from a practical point of view:

1. To model problems with logical conditions. In this case, the integrality is even reduced to variables that only take the value 0 or 1 (see the next model [examp-ip01](#)) for more information about them and how to specify logical constraints).
2. To model combinatorial problems, such as *sequencing problems*, (*job-shop scheduling*, and many others. Many of them are again transformed to integer problems with 0–1 variables. A small example can be found here: [knapsack](#).
3. To model non-linear problems. There exist techniques that translate a non-linear problem into a integer (linear) problem. (For more information see, for example [bill320](#)).
4. To model problems where we need discrete (integer) numbers for various entities. An example for this last category is given by the following problem: [magics](#) another is [mobile1](#).

Various linear problems with a special structure (the matrix A must be unimodular) such as the *transportation problem* have integer solutions without explicitly formulating them as integer programs. They can be solved as LP programs.

Models with general integer variables with an upper bound can also be transformed into models that only contain 0 – 1 integer variables. The transformation is as follows. Let $x \in \{0, 1, \dots, u\}$ be a general upper (and lower) bounded integer variable. Then substitute the variable $0 \leq x \leq u$ by the expression:

$$\delta_0 + 2\delta_1 + 4\delta_2 + \dots + 2^r\delta_r$$

where $\delta_0, \dots, \delta_r$ are 0 – 1 integer variables, and r is the smallest number, such that $u \leq 2^r$.

Furthermore, problems which contain general integer variables when modelling them in a straightforward way are sometimes preferably modeled with 0 – 1 variables. An example is the Sudoku game (see the [puzzlebook](#) that has plenty of such models).

6.5. A 0-1 Integer Program ([examp-ip01](#))

— Run LPL Code , [HTML Document](#) —

Problem: The general **linear 0-1 integer programming** model – called **0-1-IP** – consists of a linear objective function $f(x)$, m linear constraints $g_i(x)$, and n $0 - 1$ integer variables. It can be compactly formulated as follows (see [83]):

$$\begin{aligned} \min \quad & \sum_{j \in J} c_j x_j \\ \text{subject to} \quad & \sum_{j \in J} a_{i,j} x_j \geq b_i \quad \text{forall } i \in I \\ & x_j \in \{0, 1\} \quad \text{forall } j \in J \\ & I = \{1 \dots m\}, J = \{1 \dots n\}, m, n \geq 0 \end{aligned}$$

A more compact formulation using matrix notation for the model is:

$$\begin{aligned} \min \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} \quad & \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \in \{0, 1\} \end{aligned}$$

The objective function f the constraints g_i , and \mathbf{X} (in the general model format) are as follows:

$$\begin{aligned} f(x_1, \dots, x_n) &= c_1 x_1 + \dots + c_n x_n \\ g_i(x_1, \dots, x_n) &= b_i - (a_{i,1} x_1 + \dots + a_{i,n} x_n) \leq 0 \quad \text{forall } i = 1, \dots, m \\ x &\in \{0, 1\} \end{aligned}$$

The 0-1-IP model has the same notation as the LP (and the IP) model, the only difference is that the variables are $0 - 1$ integer values. The 0-1-IP model is also difficult to solve in general. (To get an idea solve the following model: [lp2000](#). Then try to solve [ip01-2000](#).)

In the following a problem with random data is specified (with $n = 15$ and $m = 15$):

$$\mathbf{c} = (16 \ 73 \ 39 \ 81 \ 68 \ 60 \ 90 \ 11 \ 33 \ 89 \ 71 \ 12 \ 24 \ 23 \ 47)$$

$$A = \begin{pmatrix} 87 & 34 & 0 & 0 & 43 & 0 & 52 & 85 & 36 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 39 & 0 & 84 & 0 & 0 & 88 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\ 55 & 63 & 79 & 0 & 0 & 0 & 0 & 71 & 0 & 0 & 0 & 0 & 0 & 70 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 73 & 0 & 0 & 0 & 22 & 0 & 0 & 79 \\ 0 & 0 & 0 & 66 & 0 & 0 & 34 & 0 & 0 & 24 & 0 & 61 & 0 & 0 & 64 \\ 0 & 0 & 16 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 15 & 70 & 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 52 & 0 & 14 & 0 & 0 & 92 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 98 & 0 & 19 & 0 & 0 \\ 0 & 70 & 37 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 86 & 0 & 93 & 0 \\ 90 & 0 & 82 & 0 & 66 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 \\ 77 & 30 & 0 & 0 & 0 & 0 & 13 & 10 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 72 & 0 & 0 & 0 & 79 & 0 & 0 & 0 & 0 & 0 & 11 & 90 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 & 55 & 0 & 23 & 0 & 0 & 0 & 0 & 0 & 36 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 58 \\ 44 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 119 \end{pmatrix}$$

The data given above specify the following explicit linear program:

$$\begin{aligned} \text{min} \quad & 16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 + 33x_9 \\ & + 89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\ \text{subject to} \quad & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\ & 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\ & 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\ & 73x_8 + 22x_{12} \geq 0 \\ & 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\ & 16x_3 + 19x_4 + 18x_{15} \geq 0 \\ & 15x_1 + 70x_2 + 61x_3 \geq 0 \\ & 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\ & 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\ & 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\ & 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\ & 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\ & 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\ & 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\ & x_j \in [0, 1] \quad \text{forall } j \in \{1 \dots 15\} \end{aligned}$$

Modeling Steps: The formulation of the model in LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First we define the two sets i and j . Then we declare and assign the data as parameters A , c , and b . The variable vector x is declared with the keyword `binary` (the only difference to the LP model), and finally the constraints R and the minimizing objective function `obj`.

Note that the data matrices **A**, **b**, and **c** are generated using LPL's own random generator. (To generate each time the same data, the code can also use the function `SetRandomSeed(a)` where `a` is an integer.) Note also that only difference in the LPL formulation compared with the LP model is the word **binary** added to the variable declaration.

Listing 6.3: The Complete Model implemented in LPL [72]

```
model Ip1501 "A 0-1 Integer Program";
  set i := [1..15]; j := [1..15];
  parameter A{i,j} := Trunc(if(Rnd(0,1)<0.25, Rnd(10,100)
    )); c{j} := Trunc(Rnd(10,100));
  b{i} := Trunc(if(Rnd(0,1)<0.15, Rnd(0,120))
    );
  binary variable x{j};
  constraint R{i} : sum{j} A*x >= b;
  minimize obj : sum{j} c*x;
  Writep(obj,x);
end
```

Solution: The small model defined above has the following solution :

$$x = (0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1)$$

The optimal value of the objective function is:

$$\text{obj} = 255$$

Comparing the optimal solution of the three problems (1) `examp-lp`, (2) `examp-ip`, and (3) this one, we have the following optimal values: 174.8096 for the LP, 201 for the IP and 255 for the 0-1-IP model.

Do the increasing optimal values for LP, IP and 0-1-IP make sense? Of course, the IP model is “more” restricted, it only can have integer values, hence the IP optimal value can never be smaller than the LP optimal value. The same is true for the 0-1-IP optimum compared with the IP optimum.

One may take the bait to solve the 0-1-IP problem using the following procedure :

1. Replace the requirement that the variables are 0 – 1 integer variables by the constraint $0 \leq x \leq 1$, then solve this LP problem. (This LP problem is called the **LP relaxation** of the 0-1 integer problem.)

2. All solution values for x are in the interval $[0 \dots 1]$.
3. Finally, round their values up to 1 or down to 0, depending of whether the value is closer to 1 or to 0.

Voilà! We can show with a tiny example (see [willi155z](#)), that this apparently reasonable approach is completely erroneous: while the LP relaxation of this tiny example has a feasible solution, the corresponding 0-1 problem is infeasible.

It seems difficult to derive the integer solution from the continuous LP problem. For a systematic procedure – called **cutting plane method** – that starts with the continuous LP problem to find an integer solution of the 0-1-IP problem, see some explanation in the model example [examp-ip01r](#).

Further Comments: There is a surprisingly rich application field for 0-1-integer programming, as there is for integer programming in general. 0-1 integer programming is used in the following context:

1. To model problems with logical conditions, Boolean constraints or expressing some kind of “dichotomy”.
2. To model combinatorial problems, such as *sequencing problems* and others.
3. To model non-linear problems. They can often be translated into 0-1 integer (linear) problems.

For a short guide to 0-1 integer model formulation and how logical conditions can be integrated into a mathematical model see the paper [\[68\]](#).

6.6. An LP-relaxation of the 0-1 Program ([examp-ip01r](#))

— Run LPL Code , [HTML Document](#) —

Problem: This model is the same as the model [examp-ip01](#) with the important difference that the variables are continuous and bounded by the interval [0..1]. Hence, this model is a LP program with continuous variable. It is called the **LP relaxation** of the corresponding 0-1 integer program.

A compact formulation using matrix notation for the model is:

$$\begin{array}{ll} \min & c \cdot x \\ \text{subject to} & A \cdot x \geq b \\ & 0 \leq x \leq 1 \end{array}$$

With the same data as in model [examp-ip01](#), we get the following model:

$$\begin{array}{ll} \min & 16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 + 33x_9 \\ & + 89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\ \text{subject to} & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\ & 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\ & 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\ & 73x_8 + 22x_{12} \geq 0 \\ & 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\ & 16x_3 + 19x_4 + 18x_{15} \geq 0 \\ & 15x_1 + 70x_2 + 61x_3 \geq 0 \\ & 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\ & 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\ & 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\ & 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\ & 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\ & 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\ & 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\ & 0 \leq x_j \leq 1 \quad \text{forall } j \in \{1 \dots 15\} \end{array}$$

Modeling Steps: The formulation of the model in LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First we define the two sets *i* and *j*. Then we declare and assign the data as parameters *A*, *c*, and *b*. The variable vector *x* is declared, and finally the constraints *R* and the minimizing objective function *obj*. Note that the only difference between this model and the model [examp-ip01](#) is the variable declaration. The keyword `binary` has been removed and a lower and upper bound value for the variable [0..1] has been added.

Note that the data matrices **A**, **b**, and **c** are generated using LPL's own random generator. (To generate each time the same data, the code can also use the function SetRandomSeed (a) where a is an integer.)

Listing 6.4: The Complete Model implemented in LPL [72]

```

model Ip15_01r "An LP-relaxation of the 0-1 Program";
  set i := [1..15]; j := [1..15];
  parameter A{i,j} := Trunc(if(Rnd(0,1)<0.25, Rnd(10,100)
    );
    c{j} := Trunc(Rnd(10,100));
    b{i} := Trunc(if(Rnd(0,1)<0.15, Rnd(0,120))
      );
    X{j} := [0 0 0 0 0 1 1 1 0 0 0 0 1 1 1];
    // X = 0-1 solution of the examp-ip01.
    lpl model
  variable x{j} [0..1];
  constraint R{i} : sum{j} A*x >= b;
  --ADDED_a1: x[11]+x[14] >= 1; //add constraints
  --ADDED_a2: x[9] +x[14] >= 1;
  --ADDED_b1: x[13]+x[11] >= 1;
  --ADDED_b2: x[8] +x[11] >= 1;
  --ADDED_c: x[6] = 1;
  --ADDED_d: x[7] = 1;
  --ADDED_e: x[15] = 1;
  minimize obj : sum{j} c*x;
  Write('The optimal solution is as follows:\n
        Obj value = %8.4f , %9d , %9d
                    rounded true 0-1 value\n',
        obj, sum{j} c*Round(x), sum{j} c*X);
  Write{j} ('           x(%2s) = %8.4f %5d %10d\n', j,x,
    Round(x),X);
end

```

Solution: The model has the following solution:

$$x = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0.16 & 1 & 1 & 1 & 0 \\ 0.14 & 0 & 0 & 0.04 & 1 & & & & & \end{pmatrix}$$

The optimal value of the objective function is:

$$\text{obj} = 201.52$$

In the following listing, we compare the three solutions: (1) the LP relaxation, (2) the rounded solution of the LP relaxation, and (3) the 0-1-IP solution. The LP relaxation has the optimal solution of 201.5179, the rounded problem has a solution of 181, which is far away from the true 0-1 solution which is 255. The rounded solution has no merit for the true integer solution.

Obj value = 201.5179 ,	181 ,	255
	rounded	true 0-1 value
x(1) = 0.0000	0	0
x(2) = 0.0000	0	0
x(3) = 0.0000	0	0
x(4) = 0.0000	0	0
x(5) = 0.0000	0	0
x(6) = 0.1563	0	1
x(7) = 1.0000	1	1
x(8) = 1.0000	1	1
x(9) = 1.0000	1	0
x(10) = 0.0000	0	0
x(11) = 0.1429	0	0
x(12) = 0.0000	0	0
x(13) = 0.0000	0	1
x(14) = 0.0435	0	1
x(15) = 1.0000	1	1

In contrast to the rounded version, the LP relaxation has an important function for the integer problem: The LP relaxation *generates a lower bound* for the integer solution. By solving the LP relaxation, we know that the optimal value of the integer problem *cannot be below* the optimal value of the LP relaxation. In our model, the optimal solution of the integer problem must be at least 201.5179 (we know already that it is 255). This is an important fact.

But we may say more about the relation between the LP relaxation and the 0-1-IP problem. For example, we can look at a particular inequality. Let's choose just an arbitrary one, say:

$$52x_9 + 14x_{11} + 92x_{14} \geq 58$$

What can we say about that particular inequality? If x_{14} is zero then both x_9 and x_{11} must be one. Why? Because if $x_{14} = 0$, then the inequality reduces to:

$$52x_9 + 14x_{11} \geq 58$$

However, this can only be the case if – in the 0-1 integer problem – both x_9 and x_{11} are 1. Hence, we can *add* the two following inequalities to the LP relaxation model:

$$x_{14} + x_9 \geq 1 , \quad x_{14} + x_{11} \geq 1$$

Why? These two additional constraints do *not* violate the 0-1-IP solution: if $x_{14} = 1$ then both inequalities are fulfilled, if $x_{14} = 0$ then both x_9 and x_{11}

must be 1. That is exactly what the initial inequality claims if the values must be 0 or 1.

Now we solve the problem again without excluding a feasible solution of the 0-1 IP problem. What is interesting now: After having added these two constraints to the LP relaxation and solving it again, the optimal solution will be 220.2321. It has increased considerably, and again we can say that this is a lower bound for the 0-1 integer problem.

In the same way we could now look at the inequality:

$$30x_8 + 98x_{11} + 19x_{13} \geq 44$$

and we can repeat the same idea: If $x_{11} = 0$ then both x_8 and x_{13} must be 1 in the integer program. This gives rise to the additional inequalities:

$$x_{11} + x_8 \geq 1 \quad , \quad x_{11} + x_{13} \geq 1$$

Adding them too to the LP relaxation and solving the problem in \mathbf{R}^{15} , gives a optimal solution of 237.3750. That again rises the lower bound for the integer program considerably.

Looking at the solution, the unique value that is not integer is $x_6 = 0.1563$. In the integer problem, x_6 must be 0 or 1. So let try to set $x_6 = 0$ and add this to the previous problem. Solving the problem results in an infeasible problem. Hence, there is no integer solution where $x_6 = 0$. So let's try $x_6 = 1$ instead. We add this inequality to the previous problem. The new optimal solution is 243.8182.

Again, in the new solution we see that $x_7 = 0.5091$, the unique value that is not integer. Hence, we try the same procedure again: setting first $x_7 = 0$ and solving produces also an infeasible solution, setting $x_7 = 1$, gives an optimal solution of 249.7778.

There is still one variable that is not integer: $x_{15} = 0.8889$. Adding $x_{15} = 0$ gives an infeasible solution, but setting $x_{15} = 1$ produces an integer solution with the optimum of 255. This is identical to the 0-1-IP problem and we have found the optimal solution to the 0-1-IP problem by adding appropriate inequalities (and equalities) to the LP relaxation. In our case, we added 4 inequalities and three equalities (by setting three variables to 1) (see the commented lines --ADDED... in the LPL code).

We conclude that the LP relaxation is important for the integer solution. It is the starting point of an iterative procedure that adds “valid” inequalities for

the integer problem, until eventually we reach the optimal point of the integer problem. (Unfortunately, it is normally not so easy to add valid inequalities.) At least we have sketched an interesting idea on how to attack the solution of integer problems, that has great practical importance. For a more systematic approach to integer programming, see the interesting book [42]. See also two other small example in this context: [alterna](#) and [unimodular](#).

6.7. A Quadratic Convex Program ([examp-qp](#))

— [Run LPL Code](#) , [HTML Document](#) —

Problem: The general **quadratic programming** model – called **QP** – consists of m linear constraints, n variables and a quadratic convex objective function $f(x)$. It can be compactly formulated as follows:

$$\begin{aligned} \min \quad & \sum_{j \in J, k \in J} x_j Q_{j,k} x_k + \sum_{j \in J} c_j x_j \\ \text{subject to} \quad & \sum_{j \in J} a_{i,j} x_j \geq b_i \quad \text{forall } i \in I \\ & x_j \geq 0 \quad \text{forall } j \in J \\ & J = \{1 \dots m\}, I = \{1 \dots n\}, m, n \geq 0 \end{aligned}$$

An more compact formulation using matrix notation for the model is as follows:

$$\begin{aligned} \min \quad & \mathbf{x} \mathbf{Q} \mathbf{x}' + \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} \quad & \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

The objective function f the constraints g_i , and X (in the general format) are as follows:

$$\begin{aligned} f(x_1, \dots, x_n) &= x_1 Q_{1,1} x_1 + x_1 Q_{1,2} x_2 + \dots + x_n Q_{n,n-1} x_{n-1} + x_n Q_{n,n} x_n \\ g_i(x_1, \dots, x_n) &= b_i - (a_{i,1} x_1 + \dots + a_{i,n} x_n) \leq 0 \quad \text{forall } i = 1, \dots, m \\ \mathbf{x} &\in \mathbb{R}^n \end{aligned}$$

Note that the matrix \mathbf{Q} must be *semi-definite positive (SDP)*, (that is: there exist a vector x such that $\mathbf{x} \mathbf{Q} \mathbf{x}' \geq 0$). In many applications the matrix \mathbf{Q} is also symmetric ($\mathbf{Q} = \mathbf{Q}^T$).

If the matrix \mathbf{Q} is not semi-definite positive then it cannot be solved as a convex problem and it must be considered as an non-linear problem.

In the following, a concrete problem with random data for the two matrices A , Q and the two vectors b and c is specified (with $n = 15$ and $m = 15$):

$$\mathbf{c} = (16 \ 73 \ 39 \ 81 \ 68 \ 60 \ 90 \ 11 \ 33 \ 89 \ 71 \ 12 \ 24 \ 23 \ 47)$$

$$A = \begin{pmatrix} 87 & 34 & 0 & 0 & 43 & 0 & 52 & 85 & 36 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 39 & 0 & 84 & 0 & 0 & 88 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\ 55 & 63 & 79 & 0 & 0 & 0 & 0 & 71 & 0 & 0 & 0 & 0 & 70 & 0 & 79 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 73 & 0 & 0 & 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 66 & 0 & 0 & 34 & 0 & 0 & 24 & 0 & 61 & 0 & 0 & 64 \\ 0 & 0 & 16 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 15 & 70 & 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 52 & 0 & 14 & 0 & 0 & 92 & 0 & 58 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 98 & 0 & 19 & 0 & 0 & 44 \\ 0 & 70 & 37 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 86 & 0 & 93 & 0 & 0 \\ 90 & 0 & 82 & 0 & 66 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 \\ 77 & 30 & 0 & 0 & 0 & 0 & 13 & 10 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 72 & 0 & 0 & 0 & 79 & 0 & 0 & 0 & 0 & 11 & 90 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 & 55 & 0 & 23 & 0 & 0 & 0 & 0 & 0 & 119 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 119 \end{pmatrix}$$

$$\text{diag}(Q) = (7 \ 5 \ 5 \ 6 \ 17 \ 19 \ 6 \ 8 \ 16 \ 19 \ 12 \ 5 \ 13 \ 17 \ 18)$$

Note that this matrix Q consisting of positive diagonal entries and zero otherwise is semidefinite positive.

The data given above specify the following explicit linear program:

$$\begin{array}{ll} \min & 7x_1^2 + 5x_2^2 + 5x_3^2 + 6x_4^2 + 17x_5^2 + 19x_6^2 + 6x_7^2 + 8x_8^2 \\ & + 16x_9^2 + 19x_{10}^2 + 12x_{11}^2 + 5x_{12}^2 + 13x_{13}^2 + 17x_{14}^2 + 18x_{15}^2 \\ & + 16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 \\ & + 33x_9 + 89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\ \text{subject to} & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\ & 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\ & 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\ & 73x_8 + 22x_{12} \geq 0 \\ & 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\ & 16x_3 + 19x_4 + 18x_{15} \geq 0 \\ & 15x_1 + 70x_2 + 61x_3 \geq 0 \\ & 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\ & 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\ & 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\ & 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\ & 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\ & 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\ & 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\ & x_j \geq 0 \quad \text{forall } j \in \{1 \dots 15\} \end{array}$$

Modeling Steps: The formulation of the model in LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First, the two sets i and j are defined. Then the data are declare and assigned as parameters A , c , b , and a semidefinite positive matrix Q . The variable vector x is declared, and finally the constraints R and the minimizing objective function obj are written.

Note that the data matrices A , b , c , and Q are generated using LPL's own random generator. (To generate each time the same data, the code can also use the function `SetRandomSeed(a)` where a is an integer.)

Listing 6.5: The Complete Model implemented in LPL [72]

```

model Qp15 "A Quadratic Convex Program";
  set i := [1..15]; j,k := [1..15];
  parameter A{i,j}:= Trunc(if(Rnd(0,1)<.25,Rnd(10,100)));
    c{j} := Trunc(Rnd(10,100));
    b{i} := Trunc(if(Rnd(0,1)<.15, Rnd(0,120)));
    Q{j,k}:= Trunc(if(j=k, Rnd(5,20))); //SDP
  variable x{j};
  constraint R{i} : sum{j} A*x >= b;
  minimize obj : sum{j} c*x + sum{j,k} x[j]*Q*x[k];
  Writep(obj,x);
end

```

Solution: The small model defined above has the following optimal solution:

$$x = (0 \ 0 \ 0 \ 0 \ 0 \ 0.051 \ 1.37 \ 0.7 \ 0.839 \ 0 \ 0.23 \ 0 \ 0 \ 0.121 \ 0.63)$$

The optimal value of the objective function is:

$$obj = 245.4168$$

Further Comments: There are interesting applications in *Portfolio Theory* for the quadratic convex problems. Especially, the Markowitz approach in portfolio models can be formulated as a QP model. For several implemented models see [Casebook I](#). Other applications are robust optimization and chance constraint optimization, and much more.

Note that the constraints can also be quadratic, we then have a QPC model where the Q matrix must be semi-definite positive:

$$\begin{aligned} \min \quad & \sum_{j \in J} c_j x_j \\ \text{subject to} \quad & \sum_{\substack{j \in J, k \in J}} x_j Q_{j,k} x_k + \sum_{j \in J} a_{i,j} x_j \geq b_i \quad \text{forall } i \in I \\ & x_j \geq 0 \quad \text{forall } j \in J \\ & J = \{1 \dots m\}, I = \{1 \dots n\}, m, n \geq 0 \end{aligned}$$

6.8. A 0-1-Quadratic Program ([examp-qp01](#))

— Run LPL Code , HTML Document –

Problem: The general **0-1-quadratic (convex) programming** model – called **0-1-QP** – contains n linear constraints and m binary variables and a quadratic convex objective function. It can be compactly formulated as follows (see [83]):

$$\begin{aligned} \min \quad & \sum_{j \in J, k \in J} x_j Q_{j,k} x_k + \sum_{j \in J} c_j x_j \\ \text{subject to} \quad & \sum_{j \in J} A_{i,j} \cdot x_j \geq b_i \quad \text{forall } i \in I \\ & x_j \in \{0, 1\} \quad \text{forall } j \in J \\ & J = \{1 \dots m\}, I = \{1 \dots n\}, m, n \geq 0 \end{aligned}$$

An even more compact formulation using matrix notation for the model is :

$$\begin{aligned} \min \quad & x \cdot Q \cdot x' + c \cdot x \\ \text{subject to} \quad & A \cdot x \geq b \\ & x \in \{0, 1\} \end{aligned}$$

Solve this problem – x are unknowns, A , b , and c are given – with $n = 15$ and $m = 15$ with the following data:

$$c = (16 \ 73 \ 39 \ 81 \ 68 \ 60 \ 90 \ 11 \ 33 \ 89 \ 71 \ 12 \ 24 \ 23 \ 47)$$

$$A = \begin{pmatrix} 87 & 34 & 0 & 0 & 43 & 0 & 52 & 85 & 36 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 39 & 0 & 84 & 0 & 0 & 88 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\ 55 & 63 & 79 & 0 & 0 & 0 & 0 & 71 & 0 & 0 & 0 & 0 & 70 & 0 & 79 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 73 & 0 & 0 & 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 66 & 0 & 0 & 34 & 0 & 0 & 24 & 0 & 61 & 0 & 0 & 64 \\ 0 & 0 & 16 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 15 & 70 & 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 52 & 0 & 14 & 0 & 0 & 92 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 98 & 0 & 19 & 0 & 0 \\ 0 & 70 & 37 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 86 & 0 & 93 & 0 \\ 90 & 0 & 82 & 0 & 66 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 \\ 77 & 30 & 0 & 0 & 0 & 0 & 13 & 10 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 72 & 0 & 0 & 0 & 79 & 0 & 0 & 0 & 0 & 0 & 11 & 90 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 & 55 & 0 & 23 & 0 & 0 & 0 & 0 & 36 & 119 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 58 \\ 44 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 119 \end{pmatrix}$$

$$\text{diag}(Q) = (7 \ 5 \ 5 \ 6 \ 17 \ 19 \ 6 \ 8 \ 16 \ 19 \ 12 \ 5 \ 13 \ 17 \ 18)$$

The data given above specify the following explicit linear program:

$$\begin{aligned}
 \min \quad & 7x_1x_1 + 5x_2x_2 + 5x_3x_3 + 6x_4x_4 + 17x_5x_5 + 19x_6x_6 \\
 & + 6x_7x_7 + 8x_8x_8 + 16x_9x_9 + 19x_{10}x_{10} + 12x_{11}x_{11} \\
 & + 5x_{12}x_{12} + 13x_{13}x_{13} + 17x_{14}x_{14} + 18x_{15}x_{15} \\
 & + 16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 \\
 & + 33x_9 + 89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\
 \text{subject to } & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\
 & 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\
 & 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\
 & 73x_8 + 22x_{12} \geq 0 \\
 & 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\
 & 16x_3 + 19x_4 + 18x_{15} \geq 0 \\
 & 15x_1 + 70x_2 + 61x_3 \geq 0 \\
 & 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\
 & 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\
 & 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\
 & 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\
 & 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\
 & 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\
 & 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\
 & x_j \in \{0, 1\} \quad \text{forall } j \in \{1 \dots 15\}
 \end{aligned}$$

Modeling Steps: A 0-1-quadratic program (0-1-QP) is a mathematical model that consists of a number ($n \geq 0$) of linear inequalities in a number ($m \geq 0$) of binary variables. Furthermore, it defines an quadratic convex objective function that is to be minimized or maximized. The 0-1-QP model has many applications in quantitative decision making. The formulation of the model in LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First we define the two sets i and j . Then we declare and assign the data as parameters A , c , b , and a semidefinite positive (SDP) matrix Q . The variable vector x is declared, and finally the constraints R and the minimizing objective function obj .

Note that the data matrices A , b , c , and Q are generated using LPL's own random generator. (To generate each time the same data, the code can also use the function `SetRandomSeed(a)` where a is an integer.) Note that the unique difference within this model and the QP model (see [examp-qp](#)) consists of the word `binary` in declaration of the variables.

Listing 6.6: The Complete Model implemented in LPL [72]

```

model Qp15_01 "A 0-1-Quadratic Program";
  set i := [1..15]; j,k := [1..15];
  parameter A{i,j}:= Trunc(if(Rnd(0,1)<.25,Rnd(10,100)));
    c{j} := Trunc(Rnd(10,100));
    b{i} := Trunc(if(Rnd(0,1)<.15,Rnd(0,120)));
    Q{j,k}:= Trunc(if(j=k, Rnd(5,20))); //SDP
  binary variable x{j};
  constraint R{i} : sum{j} A*x >= b;
  minimize obj : sum{j} c*x + sum{j,k} x[j]*Q*x[k];
  Writep(obj,x);
end

```

Solution: The model has the following solution:

$$x = (0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1)$$

The optimal value of the objective function is:

$$\text{obj} = 336$$

Further Comments: Interesting applications for the iQP model come from portfolio theory. Especially, if we want to limit the number of assets in a portfolio we must use 0-1 variables. An applications come from clustering problems. An example is [bill035](#).

6.9. Second-Order Cone (**socp1**)

— Run LPL Code , HTML Document –

Problem: A second-order cone model contains quadratic constraint of the form

$$\sum_{i \in I} x_i^2 \leq w^2$$

where x_i and w are variables. These constraints are convex and they are automatically recognized and solved by Gurobi and Cplex, for example.

A small model example is given here.

Listing 6.7: The Complete Model implemented in LPL [72]

```

model socp1 "Second-Order Cone";
set i:=[1..3]; j:=[1..5];
parameter a{i,j} :=
  [19 0 -17 21 0 , 12 21 0 0 0 , 0 12 0 0 16];
variable w; x{j};
constraint
  A{i}: sum{j} a*x = 1;
  B: sum{j} x^2 <= w^2;
minimize obj: w;
  Write(' w = %8.5f\n', w);
  Write{j}(' x%1s = %8.5f \n', j,x);
end

```

6.10. Rotated second-order Cone (**socp2**)

— Run LPL Code , [HTML Document](#) —

Problem: A rotated second-order cone model contains quadratic constraint of the form

$$\sum_{i \in I} x_i^2 \leq wv$$

where x_i , v , and w are variables. These constraints are convex and they are automatically recognized and solved by Gurobi and Cplex, for example.

A small model example is given here.

Listing 6.8: The Complete Model implemented in LPL [72]

```
model socp2 "Rotated second-order Cone";
  set i:=1..3;
  parameter c{i} := [11 7 9];
  b{i} := [ 5 6 8];
  variable x{i} [0..100]; r{i} [0..100]; k{i};
  constraint
    A: sum{i} 2*r <= 1;
    B{i}: k = Sqrt(b);
    C{i}: k^2 <= 2*x*r;
  minimize obj: sum{i} c*x;
  Writep(obj,x,r,k);
end
```

6.11. A NQCP model (**bilinear**)

— Run LPL Code , HTML Document –

Problem: All models with quadratic constraints seen so far are convex: QP, QPC, SOCP.

Models with only linear and quadratic terms which are non-convex are called NQCP models in LPL. They are much harder to solve, but Gurobi has a way to solve them (the parameter “nonconvex” must be set to 2).

A small example is this model from the [Gurobi model library](#)

Listing 6.9: The Complete Model implemented in LPL [72]

```
model bilinear "A NQCP model";
variable x; y; z;
maximize obj: x;
constraint
    A: x + y + z <= 10;
    B: x * y <= 2;           --(bilinear inequality)
    C: x * z + y * z = 1;   --(bilinear equality)
    // x, y, z non-negative (x integral in second version
    )
    Writep(obj,x,y,z);
end
```

6.12. Largest Empty Rectangle (iNCQP) (**quadrect**)

— Run LPL Code , HTML Document –

Problem: Find the largest empty rectangle in a unit square filled with random points, see Figure 6.2. This problem is from [Erwin Kalvelagen](#), see also the [Blog of Hexaly](#).

The problem is interesting for checking the solvability of a *non-convex quadratic problem* using the commercial solvers [Gurobi](#) , [Cplex](#), and [Hexaly](#).

A extension to higher dimension of this model is given in [quadrect1](#).

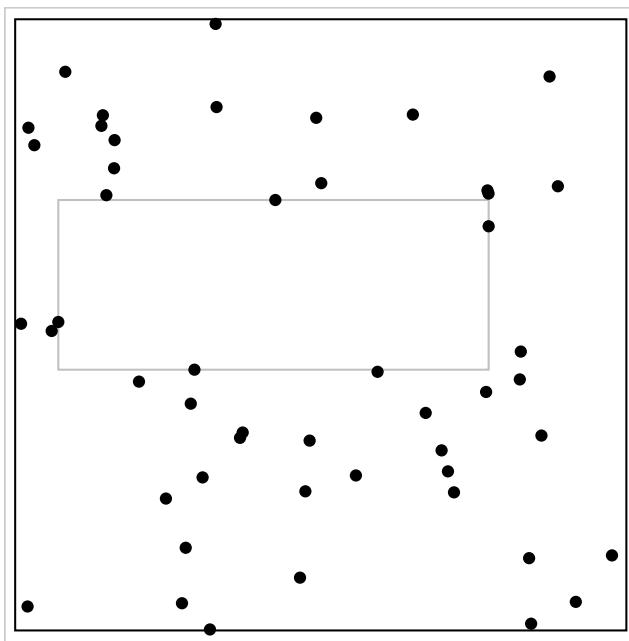


Figure 6.2: Largest Empty Rectangle with 50 points

Modeling Steps: Let us generate n points (x_i, y_i) with $i \in \{1, \dots, n\}$, that are uniformly distributed in a unit square. The unknown empty rectangle within the unit square can be defined by two corner points (top/left – bottom/right) (x^a, y^a) and (x^b, y^b) . These are the variables of our problem.

The model then can be formulated as follows³ :

³Note that the two operators \wedge and \vee are the Boolean operators “and” and “or”

$$\begin{aligned}
 \max \quad & (x^b - x^a) \cdot (y^b - y^a) \\
 \text{s.t.} \quad & x^a \leq x^b \wedge y^a \leq y^b \\
 & x^a \geq x_i \vee x^b \leq x_i \vee y^a \geq y_i \vee y^b \leq y_i \quad \text{forall } i \in \{1, \dots, n\} \\
 & 0 \leq x^a, y^a, x^b, y^b \leq 1 \\
 & n > 0
 \end{aligned}$$

The objective function defines the size of the empty rectangle (width \times height) and is a non-convex quadratic function. The first constraint requires that the point (x^a, y^a) is smaller than point (x^b, y^b) . The second constraint requires that at least one of the four disjunctive terms must be true for each i . Note, this model is a non-convex quadratic model. Recent versions of Gurobi and Cplex allow to solve such problems.

Listing 6.10: The Complete Model implemented in LPL [72]

```

model quadrect "Largest Empty Rectangle (iNCQP)";
  --SetSolver(Hexaly,'TimeLimit=100');
  --SetSolver(cplexLSol);
  set i := 1..400 "a set of points";
  parameter x{i}:=Rnd(0,1); y{i}:=Rnd(0,1);
  variable xa [0..1]; ya [0..1];
            xb [0..1]; yb [0..1];
  constraint
    A: xa<=xb and ya<=yb;
    B{i}: xa>=x or xb<=x or ya>=y or yb<=y;
  maximize obj: (xb-xa)*(yb-ya);
  Draw.Scale(300,300);
  Draw.Rect(0,0,1,1,1,0);
  Draw.Rect(xa,ya,xb-xa,yb-ya,1,2);
  {i} Draw.Circle(x,y,.01);
end

```

Further Comments: If the problem is solved with Gurobi or Cplex, LPL translates the problem automatically to the following model :

$$\begin{aligned}
 \max \quad & (x^b - x^a) \cdot (y^b - y^a) \\
 \text{s.t.} \quad & x^a \leq x^b \\
 & y^a \leq y^b \\
 & x^a \geq x_i \cdot \delta_i^1 \quad \text{forall } i \in \{1, \dots, n\} \\
 & y^a \geq y_i \cdot \delta_i^3 \quad \text{forall } i \in \{1, \dots, n\} \\
 & x^b \leq 1 - (1 - x_i) \cdot \delta_i^2 \quad \text{forall } i \in \{1, \dots, n\} \\
 & x^a \leq 1 - (1 - y_i) \cdot \delta_i^4 \quad \text{forall } i \in \{1, \dots, n\} \\
 & \delta_i^1 + \delta_i^2 + \delta_i^3 + \delta_i^4 \geq 1 \quad \text{forall } i \in \{1, \dots, n\} \\
 & 0 \leq x^a, y^a, x^b, y^b \leq 1 \\
 & \delta_i^1, \delta_i^2, \delta_i^3, \delta_i^4 \in [0, 1] \quad \text{forall } i \in \{1, \dots, n\} \\
 & n > 0
 \end{aligned}$$

Furthermore, LPL detects that this problem is an iNCQP (integer Non-Convex Quadratic Problem) and it adds automatically “NonConvex=2” (for the Gurobi solver) and “OptimalityTarget 3” (for the Cplex solver). If the solver Hexaly is used, none of these transformations are needed and LPL can pass the model as is.

Solution: The solution for $n = 50$ is shown in Figure 6.2 above.

With $n = 200$ (200 points), Gurobi 11.0 solves the problem within 8secs optimally, Cplex 20.0 has still a gap of 40% after 10mins and seems no to advance substantially. The Hexaly takes less than 2secs to solve it to optimality.

With $n = 400$, Gurobi found the optimum after 10mins, and the proof of optimality was found after 20mins. Hexaly found the optimum after 10secs and the proof of optimality after 20secs.

This result confirms the claim at the [Hexaly Blog](#).

Further Notes: In LPL, the second constraint could also be replaced by:

```

binary variable d1{i}; d2{i}; d3{i}; d4{i};
constraint
  B1{i}: d1 -> x <= xa;
  B2{i}: d2 -> y <= ya;
  B3{i}: d3 -> x >= xb;
  B4{i}: d4 -> y >= yb;
  B5{i}: d1+d2+d3+d4 >= 1;
  
```

Still another formulation is the linearization as given above (this is also the version to which LPL translates the model in order to solve it with Gurobi):

```

binary variable d1{i}; d2{i}; d3{i}; d4{i};
constraint
  D1{i}: xa >= x*d1;
  D2{i}: ya >= y*d2;
  
```

```
| D3{i}: xb <= 1-(1-x)*d3;  
| D4{i}: yb <= 1-(1-y)*d4;  
| D5{i}: d1+d2+d3+d4 >= 1;
```

Questions

1. Modify the model for any dimension
2. Find the 5 smallest non-overlapping rectangles.

Answers

1. The model is implemented in [quadrect1](#).
2. An implementation is given in model [quadrect2](#).

6.13. A NLP (non-linear) Model (chain)

— Run LPL Code , HTML Document —

The class of non-linear models is too diverse to be covered, too diverse in the sense of solution methods. Only one example is given here. Implemented in LPL, it is sent to the solver **Knitro** to be solved. The following is the problem of a hanging chain.

Problem: Find the function $y = f(x)$ of a hanging chain (of uniform density) of length L suspended between the two points (x_a, y_a) and (x_b, y_b) where $(x_a < x_b$ and $L > x_b - x_a)$ with minimal potential energy. See [16] and [40].

In physics and geometry, the curve of an idealized hanging chain or cable under its own weight when supported only at its ends is called a *catenary*. Galileo mistakenly conjectured that the curve was a parabola. Later Bernoulli and others proved that it is a hyperbolic cosine. Catenaries are used in a variety of application: An inverse catenary is the ideal shape of a freestanding arch of constant thickness. (See also in Wikipedia under “Catenary”.)

Modeling Steps: The formula for potential energy is $E = ghm$ where h is the height, m is the mass and g is a gravitational constant. As a function the height h is y and the mass m is proportional to the arc length of the chain. Hence, the model is to minimize the potential energy E under the condition of the chain length L :

$$\begin{aligned} \min \quad & \int_{x_a}^{x_b} y \cdot \sqrt{1 + y'^2} \, dx \\ \text{subject to} \quad & \int_{x_a}^{x_b} \sqrt{1 + y'^2} \, dx = L \end{aligned}$$

Explanation: It is easy to see that in a small horizontal interval Δx of the function $y = f(x)$ and the corresponding Δy interval, the arc length of the function is:

$$\sqrt{\Delta x^2 + \Delta y^2} = \sqrt{\frac{\Delta x^2}{\Delta x^2} + \frac{\Delta y^2}{\Delta x^2}} = \sqrt{1 + \left(\frac{\Delta y}{\Delta x}\right)^2} \Delta x$$

With $\Delta x \rightarrow 0$ the length of the arc of the function $y = f(x)$ between x_a and x_b is then given by:

$$\int_{x_a}^{x_b} \sqrt{1 + \left(\frac{dx}{dy}\right)^2} \, dx = \int_{x_a}^{x_b} \sqrt{1 + y'^2} \, dx$$

To implement the problem in LPL, we need to discretize the problem in say n vertical small intervals of width $\Delta x = (x_b - x_a)/n$. Let $I = \{0, \dots, n\}$ and let $x_i = x_a(1 - i)/n + x_b i/n$ be the starting x-coordinate of the interval i , and let

$y_i = f(x_i)$. Then the function $y = f(x)$ in an interval i can be approximated by the mid-point in that interval :

$$\left(\frac{x_i + x_{i-1}}{2}, \frac{y_i + y_{i-1}}{2} \right) \quad \text{forall } i \in \{1, \dots, n\}$$

The derivative y' at interval i can be approximated by :

$$ydot_i = (y_i - y_{i-1})/\Delta x \quad \text{forall } i \in \{1, \dots, n\}$$

Finally the arc length is approximated by :

$$\sqrt{1 + ydot_i^2} \Delta x \quad \text{forall } i \in \{1, \dots, n\}$$

Note that the mass m is proportional to the arc length (since the chain has uniform density), and the gravitational constant does not influence the form of the function. So, it can be dropped. With this hints the model can be easily constructed as follows:

Listing 6.11: The Complete Model implemented in LPL [72]

```

model chain "A NLP (non-linear) Model";
parameter n := 50;
set i := 0..n "number of intervals for the
discretization";
parameter
L := 10 "length of the suspended chain";
xa := 0 "left x coordinate";
ya := 0 "height (y) of the chain at left";
xb := 5 "right x coordinate";
yb := 1 "height (y) of the chain at right" ;
dx := (xb-xa)/n "interval";
x{i} := xa+dx*i "x-coor of chain";
variable
y{i} "hanging height";
ydot{i} "derivative of y";
constraint
YDOT{i|i>0}: y[i] - y[i-1] = dx*ydot[i];
LE: sum{i|i>0} Sqrt(1+ydot[i]^2)*dx = L;
ST: y[0]=ya and y[n]=yb;
minimize energy: sum{i|i>0} (y[i]+y[i-1])/2 * Sqrt(1+
ydot[i]^2)*dx;
// 
model output;
Draw.Scale(1,10);
Draw.DefFont('Verdana',8);
Draw.XY(x,y);
end
```

```
// output the graph
model output1;
set k:=0..10;
parameter X:=1; Y:=10; --scale
x1:=400/X; y1:=300/Y; -- draw rect (0,0) to (x1,y1)
xmin:=0; xmax:=xb; ymin:=-5; ymax:=0;
Xt{k}:=xmin+(xmax-xmin)/(#k-1)*k; Yt{k}:=ymin+(ymax-ymin)/(#k-1)*k;;
x{i}:=x*x1/Abs(xmax-xmin); y{i}:=y*y1/- (ymax-ymin);
Draw.Scale(X,Y);
Draw.DefFont('Verdana',8);
Draw.Line(0,0,0,y1);
{k} Draw.Line(x1/10*k,y1,x1/10*k,y1+10/Y);
{k} Draw.Text(Round(Xt,-1)&',x1/10*k-7/X,y1+20/Y);
Draw.Line(0,y1,x1,y1);
{k} Draw.Line(-10/X,y1-y1/10*k,0,y1-y1/10*k);
{k} Draw.Text(Round(Yt,-1)&',-25/X,y1-y1/10*k+3/Y);
{i|i>0} Draw.Line(x[i-1],y[i-1],x[i],y[i]);
end
end
```

Solution: The hanging chain of length $L = 10$ between the points $(x_a, y_a) = (0, 0)$ and $(x_b, y_b) = (5, 0)$ is drawn in figure 6.3. This figure is generated by the model output.

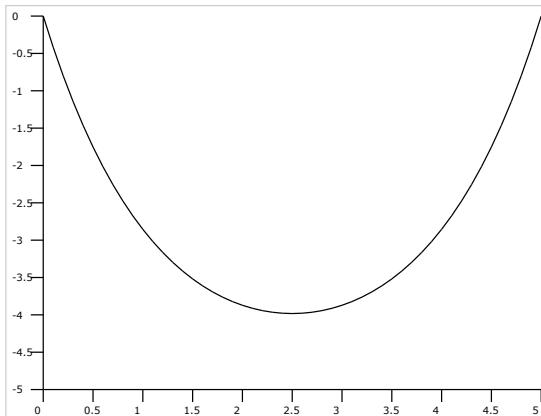


Figure 6.3: A Hanging Chain

6.14. Discrete Dynamic System (**foxrabbit**)

— Run LPL Code , HTML Document –

The class of discrete and continuous model system is huge. In LPL it is not possible right now to model differential systems, however an example of a discrete dynamic system is given here. It is the famous predator-prey model.

Problem: “Consider a forest containing foxes and rabbits where the foxes eat the rabbits for food. We want to examine whether the two species can survive in the long-term. A forest is a very complex ecosystem. So to simplify the model, we will use the following assumptions:

1. The only source of food for the foxes is rabbits and the only predator of the rabbits is foxes.
2. Without rabbits present, foxes would die out.
3. Without foxes present, the population of rabbits would grow.
4. The presence of rabbits increases the rate at which the population of foxes grows.
5. The presence of foxes decreases the rate at which the population of rabbits grows.”

(This problem is from [2] p. 132ff).

Modeling Steps: The populations are modeled using a discrete dynamical system, that is, at each point in time – let's say each month – the size of the two populations is calculated and then mapped on a graph (for more information see [2]). We calculate the populations over a period of 500 months. Hence, let $i \in I = \{0, \dots, 500\}$ a set of time points (first of month). Furthermore, let R_i and F_i be the population at time i . In the absense of the other specie, the populations would grow/shrink proportionally to the actual size, so (with $0 \leq \alpha, \delta \leq 1$) (The foxes would die out, while the population of rabbits grows infinitely.):

$$\begin{aligned}\Delta F_i &= F_{i+1} - F_i &= -\alpha F_i \\ \Delta R_i &= R_{i+1} - R_i &= \delta R_i\end{aligned}$$

Now, The presence of the other specie either increases (the foxes) its population by a rate of β , or decreases (the rabbits) its population by a rate of γ . Therefore we have:

$$\begin{aligned}\Delta F_i &= F_{i+1} - F_i &= -\alpha F_i + \beta R_i \\ \Delta R_i &= R_{i+1} - R_i &= -\gamma F_i + \delta R_i\end{aligned}$$

Reassigning the terms gives:

$$\begin{aligned} F_{i+1} &= (1 - \alpha)F_i + \beta R_i \\ R_{i+1} &= -\gamma F_i + (1 + \delta)R_i \end{aligned}$$

Given an initial population of $F_0 = 110$ and $R_0 = 900$, and the rates $\alpha = 0.12$, $\beta = 0.0001$, $\gamma = -0.0003$, and $\delta = 0.039$, it is easy to calculate the population at each time point using the simple LPL model :

Listing 6.12: The Complete Model implemented in LPL [72]

```
model foxrabbit "Discrete Dynamic System";
parameter n:=500;
set i:= 0..n;
parameter x{i}:=i; F{i}; R{i};;
F[0]:= 110, R[0]:=900,
{i|i<#i-1} {F[i+1]:=0.88*F[i] + 0.0001*F[i]*R[i],
R[i+1]:=-0.0003*F[i]*R[i] + 1.039*R[i]};
Draw.Scale(1,1);
Draw.DefFont('Verdana',8);
Draw.XY(x,F,R);
end
```

Solution: With the rates given, the populations oscillate as can be seen in Figure 6.4 (Black the population of foxes, red the population of rabbits.)

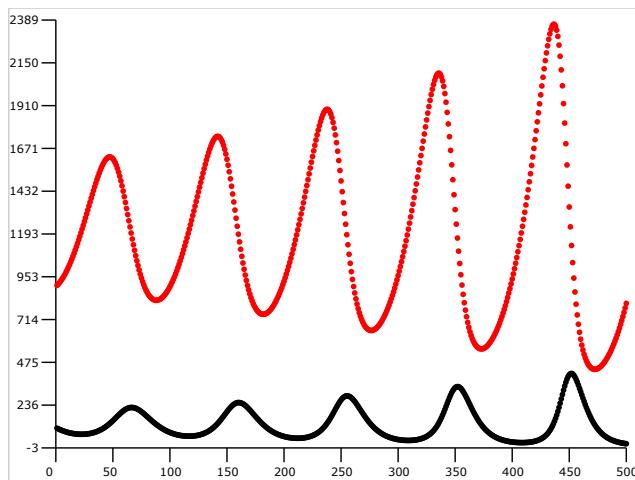


Figure 6.4: Fox-Rabbit Population over time

6.15. A Simple Permutation Model (`examp-tsp`)

— Run LPL Code , HTML Document –

An interesting model class are the permutation problems (PERM). There is a separate paper about permutation problems (see [71]), that explains in-depth what I mean by this problem class. Many of these problems can be formulated in linear (MIP) or non-linear discrete models too, but are inefficient in solving so. In simple permutation problem we are looking for a permutation out of all permutations that minimizes an expression. A typical example is the TSP problem (see also model `tsp`): The cities to be visited are numbered from 1 to n . Then any permutation of these numbers can be interpreted as “round trip” – visiting all cities in the order of the sequence of these numbers. The goal is now to find a particular permutation that minimizes the total distance. (In the [Casebook I](#), several formulations of the TSP problem are given, for example.) In practice such problems are often solved using (meta)-heuristics. There are many problems that fall into this class: quadratic assignment (QAP), flowshop scheduling, linear ordering problem (LOP), etc. As an example, the traveling salesman problem (TSP) is an used here.

Problem: Given a complete graph $G = (V, E, c)$, with a set of nodes $i, j \in V = \{1, \dots, n\}$ and edge list $E = V \times V$ and an edge length $c_{i,j}$ for each edge $(i, j) \in E$, find a *Hamilton cycle* of the graph, that is, a “round trip” in the graph that visits all nodes exactly once with minimal length (the sum of its edge lengths).

In a complete graph it is easy to get a Hamilton cycle: Every permutation of the nodes determines a Hamilton cycle. The difficulty arises when we ask for the shortest cycle. At the present time, we do not known a better method (in the worst case) to get the shortest one, then to enumerate *all* Hamilton cycles and pick the shortest, that means to check all $(n - 1)!$ permutations and check each time its length. For a small graph with $n = 30$, we already have the gigantic number of permutations of $29! = 8.84 \cdot 10^{30}$, far to big to be enumerated even by supercomputers in reasonable time. Nevertheless clever methods and mathematical models have been invented that can solve larger problems *most of the time*.

Modeling Steps: A mathematical formulation can be derived directly from the definition: Out of all permutation, find the one that minimizes the cycle.

1. Let Π be the set of all permutations, and let π be a single permutation ($\pi \in \Pi$). For example, $\pi = [1, 2, 3, \dots, n]$ is a single permutation. Let π_i be the i -th element in π .
2. Then the distance of a roundtrip of the permutation $\pi = [\pi_1, \pi_2, \dots, \pi_n, \pi_1]$ can be formulated as :

$$\sum_{i=1}^n c_{\pi_i, \pi_j}, \text{ with } j = i \bmod n + 1$$

or

$$\sum_{i=1}^{n-1} c_{\pi_i, \pi_{i+1}} + c_{\pi_n, \pi_1}$$

(j is the next element of i in the permutation π , and the next to the last is the first element in π). That is:

$$j = \begin{cases} i+1 & : i < n \\ 1 & : i = n \end{cases}$$

3. In other words: Let π be a permutation $\pi_1 \pi_2 \pi_3 \dots \pi_{n-1} \pi_n$. Then c_{π_i, π_j} with $1 \leq i, j \leq n$ is the cost from the node π_i to node π_j . The summation, therefore, expresses the costs of a round trip: $c_{\pi_1, \pi_2} + c_{\pi_2, \pi_3} + \dots + c_{\pi_{n-1}, \pi_n} + c_{\pi_n, \pi_1}$. Each possible round trip can be generated by a permutation. Minimizing this sum over all permutations means to look for the shortest round trip. Hence, the traveling salesman problem can be formulated as following:

$$\begin{aligned} \min \quad & \sum_{i=2}^n c_{\pi_{i-1}, \pi_i} + c_{\pi_n, \pi_1} \\ \text{subject to} \quad & \pi \in \Pi \end{aligned}$$

Note the syntax characteristic for this kind of models is, that (integer) variable are used as indexes: π_i is a variable. For example, $\pi_2 = 7$ means that 7 is at the second position (from left to right) in the permutation. And c_{π_i, π_j} is the distance value of $c_{h,k}$ where $h = \pi_i$ and $k = \pi_j$. This notation extension allows the modeler to formulate the problem directly in the modeling language LPL:

Listing 6.13: The Main Model implemented in LPL [72]

```
model tsp "A Simple Permutation Model";
  set i,j           "the node set";
  parameter c{i,j} "distance matrix";
  alldiff variable z{i} [1..#i] "a permutation";
  minimize obj: sum{i} c[z[if(i=1,#i,i-1)],z[i]];
end
```

Further Comments: The LPL code is an one-to-one formulation of the model above. The variable definition `alldiff` defines a permutation of numbers starting with 1. The minimization function is also close to the mathematical notation: in LPL, the construct with `if` is used instead of the modulo operation. One could also use the modulo operation to specify the objective function as follows:

| `minimize obj: sum{i} c[z[i],z[i%#i+1]];`

This is a very compact formulation, but how is the problem solved? A model in LPL consisting of only a permutation variable and an objective function is identified by LPL as a special model type, called here as *Simple Permutation Problem* (PERM). These problems are sent to an special solver integrated in LPL which is based on an Tabu-Search metaheuristic method⁴. A much more powerful solver is the commercial [Hexaly](#).

Solution: A random instance with 30 locations (nodes) is generated in the data model. So, $n = 30$ coordinates are generated for the nodes in a rectangle and calculate the distances as Euclidean distances:

Listing 6.14: The Data Model

```
model data;
  parameter n:=[30] "problem size";
  parameter X{i}; Y{i}; m:=Trunc(Sqrt(n));
  i:=1..n;
  X{i}:=(i%m+1)*2+Trunc(Rnd(0,2));
  Y{i}:=(i/m+1)*2+Trunc(Rnd(0,2));
  c{i,j}:= Sqrt((X[j]-X[i])^2+(Y[j]-Y[i])^2);
end
```

With the problem size $n = 30$, the Tabu-Search method normally finds the optimal solution in a few seconds, which is 51.8052 in this case (see Figure 6.5).

⁴The solver is quite primitive an its neighborhood structure is based on a 2-opt exchange. It was implemented only to demonstrate the feasibility of connecting LPL to that kind of solver – also heuristics. For a deeper insight into Metaheuristics etc. consult the Internet.

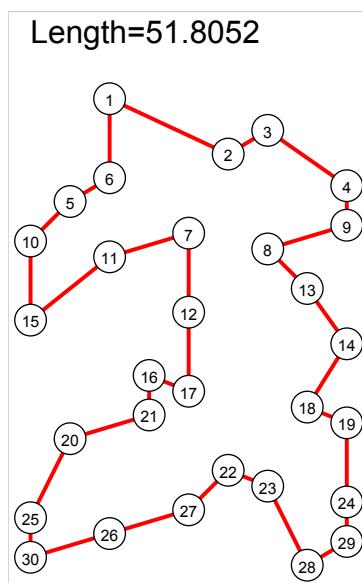


Figure 6.5: The Optimal Solution of a 30 Location Problem

6.16. Capacitated Vehicle Routing Problem ([examp-cvrp](#))

— Run LPL Code , [HTML Document](#) —

Another example of a permutation problem is the capacitated vehicle routing problem (cvrp). In this problem, we are looking for a “partitioned permutation”. A partitioned permutation is a permutation partitioned into subsequences. For example, the permutation {2, 3, 4, 1, 9, 8, 5, 6, 7, 10} could be partitioned into 3 subsequences as follows:

$$\{\{2, 3, 4\}, \{1, 9, 8, 5\}, \{6, 7, 10\}\}$$

The first subsequence is {2, 3, 4}, the second is {1, 9, 8, 5}, etc. this is exactly what is needed for the capacitated vehicle routing problem.

Problem: A fixed fleet of delivery vehicles of uniform capacity must service known customer demands for a single commodity from a common depot at minimum transit cost. An concrete application is given in [cvrp](#). The models [cvrp-1](#), [cvrp-2](#), and [cvrp-3](#) give MIP-formulations of the problem. These formulations are explained in my book [Case Studies I](#).

To interpret this problem as a “partitioned permutation” problem is easy: Starting at a depot, the first vehicle visits the customers in the first subsequence and returns to the depot, the second vehicle, also starting from the depot, visits the customers in the second subsequence and returns to the depot, etc. Note that the depot is not part of the sequences itself.

Modeling Steps: Let $i, j \in I = \{1, \dots, n - 1\}$ be a set of customer locations (n is the number of locations including the warehouse (or the depot) and let $k \in K = \{1, \dots, m\}$ be a set of trucks. Let the capacity of each truck be C_A , and let the demand quantity to deliver to a customer i be d_{m_i} . Furthermore, the distance between two customer i and j is $d_{i,j}$. Finally, the distance from the warehouse – the depot from where the trucks start – to each customer i is $d_{w,i}$.

The customers are enumerated with integers from 1 to $n - 1$. If there were one single truck (that must visits all customers), every permutation sequence of the numbers 1 to $n - 1$ would define a legal tour. Since we have m trucks, to each truck a sequence of a subset of 1 to $n - 1$ must be assigned. Hence, each truck starts at the depot, visiting a (disjoint) subset of customers in a given order and returns to the depot.

The variables can now be formulated as a “partitioned permutation”. In LPL, we can declare these partitioned permutation with a (sparse) permutation variable $x_{k,i} \in [1 \dots n - 1]$. For example, for the example above we have :

$$x = \begin{pmatrix} 2 & 3 & 4 & - & - & - & - & - & - \\ 1 & 9 & 8 & 5 & - & - & - & - & - \\ 6 & 7 & 10 & - & - & - & - & - & - \end{pmatrix}$$

Note that the two-dimensional table is sparse – it only contains $n - 1$ entries. The symbol “-“ (dash) means: no entry. Hence, $x_{1,1} = 2$ means that the customer 2 is visited by the truck 1 right after the depot, $x_{1,2} = 3$ means that the next customer (after 2) is 3, etc.

The unique constraint that must hold for the partitioned permutation is the capacity of the trucks: each subset sequence must be chosen in such a way that the capacity of the truck is larger than the cumulated demand of the customers that it visits – note, the variables $x_{k,i}$ are used as indexes as well as in the condition of the summation:

$$\sum_{i|x_{k,i}} \text{dem}_{x_{k,i}} \leq CA \quad \text{forall } k \in K$$

We want to minimize the total travel distance of the trucks. Let cc_k be the size of the subset k (the numbers of customers that the truck k visits). (In the example above $c_k = \{3, 4, 3\}$.) Of course, these numbers is variable and cannot be given in advance! Let routeDist_k be the (unknown) travel distance of truck k , then we want to minimize the total distances:

$$\min \sum_k \text{routeDist}_k$$

where⁵

$$\text{routeDist}_k = \sum_{i \in 2..cc_k} d_{x_{k,i-1}, x_{k,i}} + \text{if}(cc_k > 0, dw_{x_{k,1}} + dw_{x_{k,cc_k}}, 0) \quad \text{forall } k \in K$$

The term $dw_{x_{k,1}}$ denotes the distance of the truck k from the depot to the first customer, and $dw_{x_{k,cc_k}}$ is the distance of the truck tour k from the last customer to the depot, $d_{x_{k,i-1}, x_{k,i}}$ is the distance from a customer to the next, and $\sum_{i \in 2..cc_k} \dots$ sums that distances of a tour k .

As a variant we may also minimize, in a first round, the number of trucks used – this is added as a comment in the code:

Listing 6.15: The Main Model implemented in LPL [72]

```
model cvrp "Capacitated Vehicle Routing Problem";
```

⁵The expression `if(boolExpr, Expr)` returns `Expr` if `boolExpr` is true else it returns 0 (zero).

```

set i, j "customers";
      k "trucks";
parameter
  d{i,j} "distances";
  dw{i} "distance from/to the warehouse";
  dem{i} "demand";
  CA "truck capacity";
alldiff x{k,i} 'partition';
expression
  cc{k}: count{i} x;
  trucksUsed{k}: cc[k] > 0;
  routeDist{k}: sum{i in 2..cc} d[x[k,i-1],x[k,i]]
    + if(cc[k]>0, dw[x[k,1]] + dw[x[k,cc[k]]]);
  nbTrucksUsed: sum{k} trucksUsed[k];
  totalDistance: sum{k} routeDist[k];
constraint CAP{k}: sum{i|x} dem[x[k,i]] <= CA;
--minimize obj1: nbTrucksUsed;
minimize obj2: totalDistance;
end

```

Further Comments: The question is now what solver can solve this kind of model! I developed a simple Tabu-search solver that can deliver near optimal solution for small simple permutation problems, but not for partitioned permutation problems. However, there is a powerful commercial solver called **Hexaly** that can find near optimal solutions to large problems. LPL contains an (experimental) interface to that solver.

Solution: The LPL data code reads the “A-n32-k5.vrp” file from the Augerat et al. Set A instances.

Listing 6.16: The Data Model

```

model data;
  set h; //h is i plus 1, 1 is depot (warehouse)
  parameter de{h}; n; m; X{h}; Y{h}; string typ; dum;
  Read('A-n32-k5.vrp',%1;-1:DIMENSION',dum,dum,n);
  --Read('A-n45-k6.vrp',%1;-1:DIMENSION',dum,dum,n);
  Read('%1;-1:EDGE_WEIGHT_TYPE',dum,dum,typ);
  if typ<>'EUC_2D' then Write('Only EUC_2D is supported
    \n'); return 0; end;
  Read('%1;-1:CAPACITY',dum,dum,CA);
  Read{h} ('%1:NODE_COORD_SECTION:DEMAND_SECTION', dum,X
    ,Y);
  Read{h} ('%1:DEMAND_SECTION:DEPOT_SECTION', dum,de);
  m:=Ceil(sum{h} de/CA);
  k:=1..m;
  i:=1..n-1;
  d{i,j}:=Round(Sqrt((X[i+1]-X[j+1])^2+(Y[i+1]-Y[j+1])
    ^2));

```

```

dem{i}:=de[i+1];
dw{i}:=Round(Sqrt((X[i+1]-X[1])^2+(Y[i+1]-Y[1])^2));
end

```

The Hexaly finds the optimal solution (see 6.6) after 25secs⁶

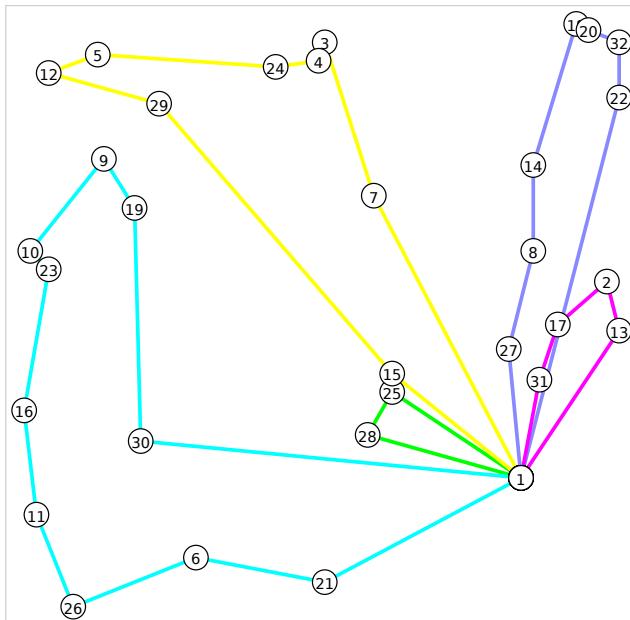


Figure 6.6: Optimal Solution to the “A-n32-k5.vrp” Instance

⁶Note that Hexaly is not simply a heuristic solver that finds hopefully near optimal solution. It integrates apparently many lower-bound checks that can guarantee the quality of the solution.

6.17. Binpacking (examp-binpack)

— Run LPL Code , HTML Document –

Problem: Another example, that can be formulated as a permutation problem is the Bin-Packing problem. In contrast to the CVRP problem (see [examp-cvrp](#)), the order of the subsets is not important: a bin just contains a subset of items, the order is not important.

The bin packing problem (BPP) is a classical model from the operations research (see also [binpack](#) for another formulation of the problem). Different items of given weights must be packed into a number of bins of given capacity. How many bins are needed to pack all items? A MIP implementation of the binpacking problem can be found in [binpack](#).

Modeling Steps: Given a set of items $i \in I$ with a weight w_i and a set of bins $k \in K$ with capacity CA (we suppose that $w_i \leq CA$ for all $i \in I$), that is all items can be packed in a bin. We need at most maxBins bins :

$$\text{maxBins} = \min(|I|, \sum_i w_i / CA)$$

So let $K = \{1, \dots, \text{maxBins}\}$.

Let's enumerate all items from 1 to $|I|$. Then we are asked to partition these numbers into maximally $|K|$ subsets (but as few as possible), such that the cumulated weight of the items in a subset do not exceed the capacity CA. The item numbers form a permutation, that must be partitioned into subsets. We can formulate this by introducing a variable $x_{k,i} \in [0 \dots |I|]$ with $k \in K$ and $i \in I$. For example: $x_{1,1} = 16$ means that the first item in bin 1 is the item with number 16, $x_{1,2} = 19$ means that the second item in bin 1 is the item with number 19, etc. $x_{i,k} = 0$ means that there is no k-th item in bin i.

The model then can be formulated as follows:

$$\begin{aligned} \text{let } & bW_k = \sum_{i|x_{k,i} \neq 0} w_{x_{k,i}} && \text{forall } k \in K \\ \text{let } & bU_k = \text{count}_i x_{k,i} > 0 && \text{forall } k \in K \\ \text{s.t. } & bW_k \leq CA && \text{forall } k \in K \\ \text{min } & \sum_k bU_k \\ \text{set partition } & x_{k,i} \in [1 \dots |I|] && \text{forall } k \in K, i \in I \end{aligned}$$

bW_k is the (unknown) weight of a bin k . bU_k is true (1) if the bin k is not empty (it counts the items in a bin k and if there is at least 1 then it is true). The unique constraint makes sure that the capacity of a bin is not exceeded, and the objective function minimizes the number of bins used.

Listing 6.17: The Complete Model implemented in LPL [72]

```

model binpacking "Binpacking";
  set i, j "items";
    k "bins";
  parameter CA "capacity of bin";
    weight{i} "weight of item i";
  alldiff x{k,i} 'set partition';
  expression bWeight{k}: sum{i|x} weight[x];
  expression bUsed{k}: count{i} x > 0;
  constraint bCapa{k}: bWeight <= CA;
  minimize nrBins: sum{k} bUsed;
  //---
  model data;
    parameter n; m;
    Read('bin-t60-00.txt',n);
    i:=1..n;
    Read('%1;1',CA);
    Read{i}('%1;2',weight);
    parameter minBins:=Ceil(sum{i} weight/CA);
      maxBins:= min(#i, 2*minBins);
    k:=1..maxBins;
  end
  //---
  model output;
    Write{k|bUsed}('Bin weight: %4d \
      | Items: %3d\n',bWeight,{i|x} x);
  end
end

```

Further Comments: In LPL, the permutation variables is declared as a **alldiff** variable with two dimensions and the attribute 'set partition' in this case.

6.17.1 Permutation Problems as Implemented in LPL

There are various kinds of permutation problems, all are declared with an **alldiff** variable in LPL. The unique commercial solver (I know) that can solve this kind of problems is the **Hexaly**.

1. The variable is one-dimensional then we are looking for a simple permutation (the keyword **variable** can be omitted):

| **alldiff variable** x{i};

Example problems are TSP, LOP (linear ordering problem), QAP, etc. If a single objective function is requested – without constraints (as in

the TSP) then the small problems can be solved with the internal Tabu-Search solver of LPL.

2. A variant of the one-dimensional permutation is the subset variant:

```
| alldiff variable x{i} 'notall';
```

In this case we are looking only for a subset of the permutation. An example is the price collecting TSP, a round-trip where not all location are visited.

3. The variable is 2-dimensional without any additional attribute then are looking for a partitioned permutation. The set i defines the permutation itself and the set k is the partitioning. The permutation is **repeated** over k .

```
| alldiff x{k, i};
```

Example with defines a permutation of 4 items repeated over 3 sets.

$$x_{k,i} = \begin{pmatrix} 2 & 3 & 4 & 1 \\ 1 & 3 & 2 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

An example is the jobshop problem, where the job sequence (permutation) is repeated over all machines (see [jobshop1](#)).

4. The variable is 2-dimensional with the additional attribute 'partition' means that the unique permutation is partitioned over the set k .

```
| alldiff x{k, i} 'partition';
```

This is the case for the [exam-crvp](#) model: the set of locations defines a permutation, which is partitioned into subtours. Note that the order in the subsets (subtours) matters.

5. The variable is 2-dimensional with the additional attribute 'set partition' means that the unique permutation is partitioned over the set k , but the order of the items in the subsets does not matter.

```
| alldiff x{k, i} 'set partition';
```

An example ist the bin-packing problem: the items to place into the bins define a permutation, which is partitioned into the bins, but the order within the bins does not matter.

6. The variable is 2-dimensional with the additional attribute 'cover' means that the unique permutation is distributed over the set k , the items in the permutation can be repeated in the different subsets.

```
| alldiff x{k, i} 'cover';
```

A example is the Split Delivery Vehicle Routing (SDVRP) problem, the same customer (location) can be visited by more than one vehicle (can be in more then one subtour) (see [sdvrp](#)).

6.18. Additional Variable Types

Variables – as seen – can be in the domain *real*, *integer*, or *binary*. An additional type of variables are special integer variables: the *permutation variables*. Three further variables types are *semi-continuous*, *semi-integer*, or *multiple-choice* variables. There is no need to introduce extra syntax for them, one can just extend the lower/upper bound specifications of the variable definitions. In LPL, the lower/upper bound are defined as [lo..up].

6.18.1 Semi-continuous Variable (**semi-1**)

— Run LPL Code , HTML Document –

A semi-continuous variable is defined as a real variable that can be zero or between a (positive) lower and upper bound.

A semi-continuous variable x can be modeled by introducing a binary variable z and an addition constraint (where lo and up are the (positive) lower and upper bounds) defined as :

$$\text{lo} \cdot z \leq x \leq \text{up} \cdot z$$

This constraint models the two alternatives: if $z = 0$ then $x = 0$, otherwise if $z = 1$ then x must be between the lower and upper bound: $\text{lo} \leq x \leq \text{up}$.

In LPL, a semi-continuous variable can simply be defined as:

```
| variable x [0,lo..up];
```

The transformations, adding a binary variable and the constraint as defined above, are done automatically by LPL

Listing 6.18: The Complete Model implemented in LPL [72]

```
model semi1 "Semi-continuous Variable";
parameter lo:=5; up:=7;
variable x [0,lo..up];
minimize obj: x;
--maximize obj: x;
Writep(x);
end
```

This model will be translated into the following model by LPL (the names may be different):

```
| model semi1 "Semi-continuous variable";
| parameter lo:=5; up:=7;
```

```

variable x;
binary variable z;
constraint A: 5*z <= x <= 7*z;
solve;
end

```

6.18.2 Semi-integer Variable (semi-2)

— Run LPL Code , HTML Document –

A semi-integer variable is defined as an integer variable that can be zero or between a (positive) lower and upper integer bound.

A semi-integer variable x can be modeled by introducing a binary variable z and an addition constraint (where lo and up are the (positive) lower and upper bounds) defined as :

$$lo \cdot z \leq x \leq up \cdot z$$

This constraint models the two alternatives: if $z = 0$ then $x = 0$, otherwise if $z = 1$ then x must be between the lower and upper bound: $lo \leq x \leq up$.

In LPL, a semi-integer variable can simply be defined as:

```
| integer variable x [0,lo..up];
```

The transformations, adding a binary variable and the constraint as defined above, are done automatically by LPL

Listing 6.19: The Complete Model implemented in LPL [72]

```

model semi2 "Semi-integer Variable";
  parameter lo:=5; up:=7;
  integer variable x [0,lo..up];
  minimize obj: x;
  --maximize obj: x;
  Writep(x);
end

```

This model will be translated into the following model in LPL (the names may be different):

```

model semi1 "Semi-integer variable";
  parameter lo:=5; up:=7;
  integer variable x;
  binary variable z;
  constraint A: 5*z <= x <= 7*z;
  solve;
end

```

6.18.3 A Multiple Choice Variable (**mchoice-3**)

— Run LPL Code , HTML Document –

A multiple choice variable is a real or integer variable, that can be assigned to values of multiple intervals. For example, a variable x is restricted to the values 2, 4, between 6 and 8, and 11, that is :

$$x \in \{2, 4, 6 \dots 8, 11\}$$

Note that a multiple-choice variable is a generalization of a semi-continuous/integer variable. It can be modeled as follows:

1. Introduce a set and two parameters as follows:

$$i \in I = \{1 \dots 4\} \quad a_i = \{2, 4, 6, 11\} \quad b_i = \{2, 4, 8, 11\}$$

2. Add a binary variable

$$z_i \in \{0, 1\} \quad i \in I$$

3. Add the two constraints:

$$\begin{aligned} \sum_{i \in I} a_i z_i &\leq x \leq \sum_{i \in I} b_i z_i \\ \sum_{i \in I} z_i &= 1 \end{aligned}$$

In LPL, the multiple-choice variable can simply be defined as (the variable may also be of type `integer`):

```
| variable x [2,4,6..8,11];
```

Listing 6.20: The Complete Model implemented in LPL [72]

```
model mchoice "A Multiple Choice Variable";
  integer variable x [2,4,6..8,11];
  minimize obj: x;
  --maximize obj: x;
  Writep(x);
end
```

This model will be translated into the following model by LPL (the names may be different):

```

model mchoice "A multiple choice variable";
  variable x;
  set i:=1..4;
  parameter a{i}:=[2,4,6,11];
              b{i}:=[2,4,8,11];
  binary variable z{i};
  constraint A: sum{i} a*z <= x <= sum{i} b*z;
                B: sum{i} z = 1;
  solve;
end

```

An alternative way to formulate a multiple-choice variable is by using the logical *or* operation. Although correct, this is not recommended in general, because it will generate more binary variables. For the example above one may formulate the model as follows:

```

model mchoice "A multiple choice variable";
  variable x [2..11];
  constraint A: x=2 or x=4 or 6<=x<=8 or x=11;
  solve;
end

```

6.19. Additional Constraint Types

Besides of linear and non-linear constraints⁷, there are many other classes in practice. Some can be transformed and formulated as linear or non-linear constraints: the *SOS1*, the *SOS2*, and the *complementarity* constraints, as explained below.

6.19.1 Sos1 Constraint (**sos-1**)

— Run LPL Code , HTML Document —

A SOS1 (special order set of type 1) constraint is a set of variables where at most one variable in the set can take a value different from zero.

Given a set of variables x_i with $i \in I = \{1, \dots, n\}$. If the variable are binary then a SOS1 constraint can be formulated simply as :

$$\sum_i x_i \leq 1$$

⁷ Since a constraint expression can include Boolean operand and Boolean operator, constraints with logical connectors are included in this definition

If the variables are real (or integer) then an additional binary variable y_i has to be introduced together with the constraints (where up_i is an upper bound on the variable x_i) :

$$\begin{aligned}\sum_i y_i &\leq 1 \\ x_i &\leq up_i \cdot y_i \quad \text{forall } i \in I\end{aligned}$$

In LPL, we can use the function `Sos1` to specify a SOS1 constraint. For example, to specify that $v = 0$ or $w = 0$ (at most one can be non-zero) we write:

```
| constraint C: Sos1(v,w)
```

To specify that only in a indexed list of variable must be non-zero, one specifies:

```
| constraint D: Sos1( {i} x[i] );
```

Listing 6.21: The Complete Model implemented in LPL [72]

```
model sos1 "Sos1 Constraint";
  set i:=1..10;
  parameter up:=10.2;
  variable x{i} [0..up];
  constraint A: Sos1({i} x);
  maximize obj: sum{i} i*x;
  Writep(obj,x);
end
```

Note only Gurobi understands this syntax, otherwise the explicit formulation must be used as follows:

```
model sos1;
  set i:=1..10;
  parameter up:=10.2;
  variable x{i} [0..up];
  binary variable y{i};
  constraint A: sum{i} y <= 1;
  constraint B{i}: x <= up*y;
  maximize obj: sum{i} i*x;
  Writep(obj,x);
end
```

6.19.2 Sos2 Constraint (`sos-2`)

— Run LPL Code , HTML Document –

A SOS2 (special order set of type 2) constraint is a set of variables where at most two adjacent variable in the set can take a value different from zero.

Given a set of variables x_i with $i \in I = \{1, \dots, n\}$. If the variables are real (or integer) then an additional binary variable y_i has to be introduced together with the constraints (where up_i is an upper bound on the variable x_i) :

$$\begin{aligned} \sum_i y_i &\leq 2 \\ x_i &\leq up_i \cdot y_i \quad \text{forall } i \in I \\ y_i + y_j &\leq 1 \quad \text{forall } i \in I, j \in \{i+2, \dots, n\} \end{aligned}$$

In LPL, we can use the function `Sos2` to specify a SOS2 constraint. For example, to specify that only in a indexed list of variable must be non-zero, one specifies:

```
| constraint D: Sos2( {i} x[i] );
```

Listing 6.22: The Complete Model implemented in LPL [72]

```
model sos2 "Sos2 Constraint";
set i:=1..10;
parameter up:=10.2;
variable x{i} [0..up];
constraint A: Sos2({i} x);
maximize obj: sum{i} i*x;
Writep(obj,x);
end
```

Note only Gurobi (and Cplex) understand this syntax from LPL, otherwise the explicit formulation must be used as follows:

```
model sos2;
set i,j:=1..10;
parameter up:=10.2;
variable x{i} [0..up];
binary y{i};
constraint B: sum{i} y <= 2;
constraint C{i}: x <= up*y;
constraint D{i,j| j>=i+2}: y[i] + y[j] <= 1;
maximize obj: sum{i} i*x;
Writep(obj,x);
end
```

6.19.3 MPEC Model Type (compl-3)

— Run LPL Code , HTML Document –

A complementarity constraint enforces that two variables are complementary to each other, that is, the following conditions hold for scalar variables x and y :

$$x \cdot y = 0 \quad , \quad x \geq 0 \quad , \quad y \geq 0$$

In LPL, this condition can be formulated by a constraint as follows:

```
| constraint A: Complements(x,y);
```

A more general version can also be implemented in LPL, where the two scalar variables can be replaced by arbitrary expressions. LPL reformulates this version. Hence, for example, the constraint

```
| constraint A1: Complements(2*x-1,4*y-1);
```

will be replaced by (where v and w are two new variables):

```
variable v; w;
constraint A1: v*w = 0;
constraint X: v = 2*x-1;
constraint Y: w = 4*y-1;
```

Note that Gurobi can solve these constraints if the expressions within the complementarity are linear or quadratic, otherwise a non-linear solver (such as Knitro) will do the job.

Listing 6.23: The Complete Model implemented in LPL [72]

```
model complement "MPEC Model Type";
  variable x [0..10]; y [0..10];
  constraint
    A1: Complements(2*x-1,4*y-1);
  maximize obj: x + y;
  Writep(obj,x,y);
end
```

6.20. Global Constraints

Global constraints are constraints representing a specific relation on a number of variables. Some of them can be rewritten as a conjunction of algebraic constraints. Other global constraints extend the expressivity of the constraint framework. In this case, they usually capture a typical structure of combinatorial problems. Global constraints are used to simplify the modeling of constraint satisfaction problems (CSP), to extend the expressivity of constraint

languages, and also to improve the constraint resolution. Many of the global constraints are referenced into an [The Online Catalog](#), a catalog that presents a list of 423 global constraints.

Actually, LPL implements only a few to illustrate the use and the power in modeling. Depending on the solver used, these constraint must be transformed.

6.20.1 Alldiff and alldiff

One of the most famous global constraint is the *alldifferent* constraint. It requires that a certain number of (integer) variables must all be different from each other. A important special case are permutations. (Permutation problems have been defined and presented in the paper [71].)

In LPL, the *alldifferent* constraint come in two versions:

1. A variable vector can be declared with the keyword **alldiff**. A typical application is the TSP problem, as formulated in **tsp**, in which the variables are formulated as a permutation.
2. A global constraint **Alldiff()** defines lists of variables that must be different from each other. Typical models are the model **sudokuM** and the model **vier**.

The first version can be used for most permutation problems, as explained in the paper [71]: for the TSP-PC (price collecting TSP), for CVRP (Capacitated Vehicle Routing Problem), Jobshop, and many others, as explained in the paper. The syntax for a simple permutation is:

```
| alldiff variable x{i} [1..#i];
```

The second version in a constraint has two forms:

```
constraint A: Alldiff(y,z,w,v);      // explicit list of
variables
constraint B: Alldiff({i} x[i]);      // indexed list of
variables
```

When using a MIP solver, LPL transforms both versions into linear constraints. There are several methods:

The first method (used by LPL) is as follows :

1. Add the binary variables: $y_{i,j}$ with $i, j \in I$.
2. Add the constraints:

$$1 \leq x_j - x_i + ny_{i,j} \leq n - 1 \quad \text{forall } i, j \in I, i < j$$

The second Method (avoiding big-M) is as follows :

1. Add the binary variables: $y_{i,j}$ with $i, j \in I$.
2. Add constraints:

$$\begin{aligned}\sum_j y_{i,j} &= 1 && \text{forall } i \in I \\ \sum_i y_{i,j} &= 1 && \text{forall } j \in I \\ x_i &= \sum_j j y_{i,j} && \text{forall } i \in I\end{aligned}$$

Note the second method can be extended. The case $x_i \in \{1, \dots, n\}$ where $i \in \{1, \dots, n\}$ is a very special case of the more general problem where $x_i \in \{a_1, \dots, a_n\}$ with $a_{i+1} > a_i$. This problem can be handled by the constraints:

$$\begin{aligned}\sum_j y_{i,j} &= 1 && \text{forall } i \in I \\ \sum_i y_{i,j} &= 1 && \text{forall } j \in I \\ x_i &= \sum_j a_j y_{i,j} && \text{forall } i \in I\end{aligned}$$

Even the case with duplicates in a ($a_{i+1} \geq a_i$ (example: $a = \{1, 2, 2, 3, 3, 3\}$)) can be handled with the previous constraints.

Another extension is to define a subset: consider $x_j \in \{a_1, \dots, a_n\}$ where $j \in \{1, \dots, m\}$ and $m < n$, for example, choose two values x_j from the set $\{1, \dots, 3\}$.

$$\begin{aligned}\sum_j y_{i,j} &\leq 1 && \text{forall } i \in I \\ \sum_i y_{i,j} &= 1 && \text{forall } j \in I \\ x_i &= \sum_j a_j y_{i,j} && \text{forall } i \in I\end{aligned}$$

For more information see [Erwin Kalvelagen](#), see also [82].

6.20.2 Element

The element constraint requires that the (unknown) x -th entry of a data vector c_i with $i \in \{1, \dots, k\}$ is exactly v .

$$\text{Element}(x, c_{i|i \in \{1, \dots, k\}}, v)$$

In an explicitly mathematical version this could be formulated as :

$$v = c_x$$

Note that x is an (integer) variable and it is at an index position. v may or may not be a variable. In LPL, the constraint can be written as follows :

```
|  constraint A: Element((x, {i}c, v);
```

When using a MIP solver the constraint is translated to :

$$\begin{aligned} v &= \sum_i c_i y_i \\ y_i &\leftrightarrow (x_i = i) \quad \text{forall } i \in I \\ y_i &\in \{0, 1\} \quad \text{forall } i \in I \end{aligned}$$

A real application model is given in model [assignCP](#).

6.20.3 Occurrence

The occurrence constraint requires that between m and n ($m \leq n$) elements of a variable vector x_i with $i \in \{1, \dots, k\}$ must be assigned the value v .

$$\text{Occurrence}(x_{i|i \in \{1, \dots, k\}}, v, m, n)$$

In an explicitly mathematical version this could be formulated as :

$$m \leq \sum_i (x_i = v) \leq n$$

Note that the term $(x_i = v)$ is a Boolean expression that returns 0 or 1, and hence, the sum counts the number of occurrences that x_i is exactly v . This number must be between m and n .

For example, let x be a variable vector of 10 elements (x_i with $i \in \{1, \dots, 10\}$), and let $v = 30$ and $m = 3$, $n = 5$. Then the constraint is fulfilled if $x = (1, 2, 5, 4, 30, 7, 30, 6, 30, 8)$ because the value 30 occurs 3 times.

In LPL, the constraint is not yet explicitly implemented. It can be formulated in the mathematical way as follows:

```
|  constraint A: m <= sum{i} (x[i] = v) <= n ;
```

When calling a MIP solver, then LPL automatically translates this into linear constraints. In the following real models this constraint is used: model [Kalis](#) and model [car2](#). In both models, the special case where $m = n$ is used, which reduces the formulation into

```
| constraint A: sum{i} (x[i] = v) = n ;
```

6.20.4 Sequence_total

The sequence_total constraint requires that in each sequence of length $s > 0$ of a variable vector x_i with $i \in \{1, \dots, k\}$ between m and n ($m \leq n$) elements must be assigned to v .

$$\text{Sequence_total}(x_{i|i \in \{1, \dots, k\}}, v, m, n, s)$$

In an explicitly mathematical version this could be formulated as :

$$m \leq \sum_{h|i \leq h \leq i+s-1} (x_i = v) \leq n \quad \text{forall } i \in \{1, \dots, k-s+1\}$$

Note that the term $(x_i = v)$ is a Boolean expression that returns 0 or 1, and hence, the sum counts the number of occurrences that x_i is exactly v . This number must be between m and n .

For example, let x be a variable vector of 10 elements (x_i with $i \in \{1, \dots, 10\}$), and let $v = 30$ and $m = 1$, $n = 2$, $s = 4$. Then the constraint is fulfilled if $x = (1, 30, 5, 4, 30, 7, 30, 6, 30, 8)$ because the value 30 occurs in each subsequence of 4 elements 1 or 2 times.

In LPL, the constraint is not yet explicitly implemented. It can be formulated in the mathematical way as follows:

```
| constraint A{i|#i-s+1}: m <= sum{h|i<=h<=i+s-1} (x[i] = v) <= n ;
```

When calling a MIP solver, then LPL automatically translates this into linear constraints. The constraint is used in the following real model of “car assembly line sequence”, see [car2](#).

6.21. Conclusion

This paper gave a limited overview of some model classes. It shows a unified implementation into the LPL modeling language. LPL can analyze the model and selects automatically the right solver, based on a classification, to solve it. The actual LPL language is far from complete, it is thought as a research vision to formulate all kind of models in a unified modeling language. That language should also contain most concepts of a modern programming language – which LPL doesn't right now.

As said, LPL is a small subset of that unified modeling language, but a special model class, namely large linear MIP models, are efficiently formulated and are used in a commercial context.

CHAPTER 7

DATA CUBES AND PIVOT-TABLES

“No gain without pain.”

A pivot table is a powerful means to represent structured, multi-dimensional data on a two-dimensional space. They enable a user to show a large amount of structured data from different angles and perspectives.

This paper gives a survey and implementation in LPL on datacube and pivot-tables in order to understand and use OLAP functionalities. It is argued that slicing, sizing, and rising are fundamental data operations of multidimensional datacubes. They are the basic building block of every OLAP tool, hence our efforts are concentrated on them. The datacube operations are explained as functions which transforms n -dimensional datacubes into datacubes of dimensions less, equal or higher than n . All operations are viewed in this perspective. This reduces various kinds of OLAP-operations considerably and looks at them in a new unified way, which also might be interesting in implementing OLAP-tools.

Then this paper exposes the connections of n -dimensional datacubes with their many 2-dimensional representations, the pivot-tables. The many different pivot-table representations of an n -dimensional datacube are also understood in a unified way, such that all representations can be generated from each other by a very limited number of operations on the datacube which could be subsumed under the general operator pivoting. It is shown how a datacube of any dimension is transformed into any pivot-table representation using just pivoting and the mentioned cube operations slicing, sizing, and rising.

It's a little-known fact... Shakespeare was working on an Excel
spreadsheet

and created a formula that inspired one of his most famous for-
mula:

=OR (B2, NOT (B2))

Spreadsheet Poem:

You may spreadsheet in columns

You may spreadsheet in rows

But the more you spreadsheet

The faster it grows.

7.1. Introduction

Data is an increasing value resource in many contexts, for example, in a company to schedule and monitor effectively the company's activities. Hence, a data manipulation system must collect and classify the data, by means of integrated and suitable procedures, in order to produce in time and at the right levels the synthesis to be used to support the decisional process, as well as to administrate and globally control the company's activity. While **databases**

are the place where data are *collected*, **data warehouses** are systems that *classify* the data. According to William Inmon, widely considered the father of the modern data warehouse, a Data Warehouse is a “ Subject-Oriented, Integrated, Time-Variant, Non-volatile collection of data in support of decision making”. Data Warehouses tend to have these distinguishing features: (1) Use a subject oriented dimensional data model; (2) Contain publishable data from potentially multiple sources and; (3) Contain integrated reporting tools.

OLAP (On-Line Analytical Processing) is a key component of data warehousing, and OLAP Services provides essential functionality for a wide array of applications ranging from reporting to advanced decision support. According to [www.olapcouncil.org] OLAP “... is a category of software technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user. OLAP functionality is characterized by dynamic multidimensional analysis of consolidated enterprise data supporting end user analytical and navigational activities including calculations and modeling applied across dimensions, through hierarchies and/or across members, trend analysis over sequential time periods, slicing subsets for on-screen viewing, drilldown to deeper levels of consolidation, rotation to new dimensional comparisons in the viewing area etc.”. The focus of OLAP tools is to provide multidimensional analysis to the underlying information. To achieve this goal, these tools employ multidimensional models for the storage and presentation of data. Data are organized in cubes (or hypercubes), which are defined over a multidimensional space, consisting of several dimensions. Each dimension comprises of a set of aggregation levels. Typical OLAP operations include the aggregation or deaggregation of information (roll-up and drill-down) along a dimension, the selection of specific parts of a cube (dicing) and the reorientation of the multidimensional view of the data on the screen (pivoting).

OLAP functionality is characterized by dynamic multi-dimensional analysis of consolidated enterprise data supporting end user analytical and navigational activities including:

- calculations and modeling applied across dimensions, through hierarchies and/or across members trend analysis over sequential time periods (data mining)
- slicing subsets for on-screen viewing
- drill-down to deeper levels of consolidation
- reach-through to underlying detail data
- rotation to new dimensional comparisons in the viewing area

Two main approaches to support OLAP are (1) ROLAP architecture (Rela-

tional On-Line Analytical Processing), and (2) MOLAP architecture (Multi-dimensional On-Line Analytical Processing). The advantage of the MOLAP architecture is, that it provides a direct multidimensional view of the data and is normally easy to use, whereas the ROLAP architecture is just a multidimensional interface to relational data and requires normally an advanced knowledge on the SQL queries. On the other hand, the ROLAP architecture has two advantages: (a) it can be easily integrated into other existing relational information systems, and (b) relational data can be stored more efficiently than multidimensional data.

Hence ROLAP is based directly on the architecture of relational databases and different vendors just extend the SQL standard language in various ways in order to implement the OLAP functionality. For example in ORACLE the cube and rollup operator expands a relational table, by computing the aggregations over all the possible subspaces created from the combinations of the attributes of such a relation. Practically, the introduced CUBE operator calculates all the marginal aggregations of the detailed data set. The value 'ALL' is used for any attribute which does not participate in the aggregation, meaning that the result is expressed with respect to all the values of this attribute.

The MOLAP architecture is basically *cube-oriented*. This does not mean that they are far from the relational paradigm in fact all of them have mappings to it but rather that their main entities are cubes and dimensions. In practice, the cube's data are extracted from databases using standard SQL queries and stored in "cubes", which beside the data extracted contain pre-calculated data in order to speed up the viewing. One of the big advantage of the cube-oriented approach is its intuitive use of generating a multitude of various "views" of the data; another advantage is speed: Having the data in a cube, it easier to reorganize the data than in ROLAP.

In this paper, the cube-oriented architecture is analyzed. In section 2 a definition of the fundamental concepts, such as datacube, is given. In section 3, the operators on datacubes are defined and analyzed. An interpretation of these operations in the light of OLAP is given in section 4. Section 5 introduces the basic concept of pivot-table, which is identified as a 2-dimensional representation of a datacube. Section 6 briefly summarizes the connection to modeling and to LPL. Finally, some reflections about spreadsheets as a modeling tool are given. See also [48], [38], [62].

7.2. Definition of Datacube

A set of $(n + 1)$ -tuples $(d_1, d_2, \dots, d_n, m)$ with $d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n, m \in M$ is called an **n -dimensional datacube**. D_1, D_2, \dots, D_n are finite sets of *members* and are the **dimensions** of the cube. M is also a set of *values* (normally numerical) called the **measurement**.

A dimension is a *structural* attribute of the cube, that is, a list of members all of which are of a similar type in the user's perception of the data. For example, all months, quarters, years, etc., make up a time dimension; likewise all cities, regions, countries, etc., make up a geography dimension. A dimension acts as an index for identifying values within a multi-dimensional array. If one member of the dimension is selected, then the remaining dimensions in which a range of members (or all members) are selected define a sub-cube. If all but two dimensions have a single member selected, the remaining two dimensions define a spreadsheet (or a "slice" or a "page"). If all dimensions have a single member selected, then a single **cell** is defined. A cell can also be identified as a single tuple of the datacube. Dimensions offer a very concise, intuitive way of organizing and selecting data for retrieval, exploration and analysis. A **member** is a discrete name or identifier used to identify a data item's position and description within a dimension. For example, *January, 1989* or *1Qtr93* are typical examples of members of a time dimension. *Wholesale, Retail*, etc., are typical examples of members of a distribution channel dimension.

As an example see the Figure 7.1.

It represents a 3-dimensional cube with three dimensions *time*, *region*, and *product*. The measurement is *Sale*. The members of time are {1996, 1997, 1998}, the members of region are {CEE, USA}, and the members of product are {P1, P2, P3}. The cell (1997, CEE, P2) contains the value 500. The interpretation is that

500 units of the product P2 has been sold in 1997 in CEE. The cube contains $3 \times 3 \times 2 = 18$ cells.

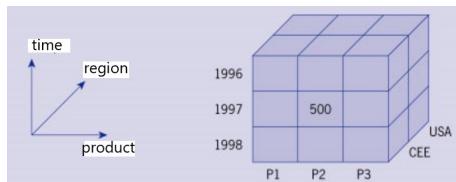


Figure 7.1: A 3-dimensional cube

A member combination is an exact description of a unique cell in the datacube which contains a single value (the measurement). A datacube can also be seen as a multi-dimensional array. A cell – a unique tuple – can be seen as a single data point that occurs at the intersection defined by selecting one member from each dimension in a multi-dimensional array. The maximal number of cells is given by the cardinalities of the dimensions as: $|D_1| \cdot |D_2| \cdot \dots \cdot |D_n|$. The tuple-set is also called the Cartesian Product of the dimensions. A datacube can be dense or sparse. It is called dense if a relatively high percentage of the possible combinations of its dimension members contain data values, otherwise it is called sparse. It is important to see, that very sparse datacubes are very common in practice.

From a database point of view, an n -dimensional datacube is typically stored as a database table containing $n+1$ fields, the first n fields representing the dimensions and the $(n+1)$ -th field represents the measurement (the data value). The first n fields are typically (but not necessary) foreign keys pointing to a

table filled with basis “identifier” lists. It is important to note that a table is a more general concept than a datacube: (1) The same tuple of members in a datacube – defining a cell – can occur only once in a datacube, while it can occur several times in a database table, except when a primary key is defined on the “dimensional” fields. In praxis, however, this is not a limitation, because we mostly analyze data which can be classified according some dimensions and hence have distinct tuples. (2) A database table – besides the “dimensional” fields – can contain several “measurements” fields. If this is the case, then several datacubes with the same dimensions can be built, or a datacube with an additional dimension, which contains a “measurement” field name as members, if the measurements are all of the same data type (numeric, alphanumeric, etc.). If this is not the case the db table can be mapped to a set of datacubes.

We call the list of tuples defining a datacube – normally printed in a vertical way – on a piece of paper, the *standard view* or the *db-view* of the datacube. As we shall see, this is nothing else than a particular pivot table view.

In a mathematical notation, a cube can be represented as follows:

$$m_{i_1, i_2, \dots, i_n} \quad \text{forall } i_1 \in D_1, i_2 \in D_2, \dots, i_n \in D_n, m \in M$$

i_1, i_2, \dots, i_n are called indexes, and the notation is called the **indexed notation** (see [67]).

7.3. Operations on Datacubes

Several operations can be applied to datacubes. The result of these operations is again a datacube. The operators are: *slicing*, *dicing*, *sizing*, and *rising*. The operation *slicing* on an n -dimensional datacube generates a datacube of dimension $\leq n$, while *dicing* and *sizing* generates a datacube of dimension n , and *rising* generates a datacube of dimension $\geq n$.

7.3.1 Slicing

Slicing means to *fix* a particular member of one or several of the dimensions (we call them “fixed” dimensions) by discarding all tuples containing any of the other members of the fixed dimensions. Then we remove the particular (fixed) members from all tuples. The result is a datacube with fewer dimensions.

Figure 7.2 shows a 3-dimensional cube. The left part selects a slice – the grey part – of a 2-dimensional cube by fixing the member 3 of dimension 1. The

right part selects a slice of 1-dimensional cube by fixing the member **3** of dimension 1 *and* the member **1** of dimension 3. In both cases, the dimension is reduced and the result is another datacube of lower dimension.

In fact, the operation “slicing on more than one dimensions” can be reduced to a sequence of “slicing on a single dimension”. The operation is commutative and associative on the dimensions, meaning that the order which it is applied to the “fixed” dimensions does not matter. “Slicing on a single dimension” reduced the dimension by one. Hence, applying this sliding operation to an n -dimensional cube, generates a $(n - 1)$ -dimensional cube. One can apply this operation at most n times successively to get a 0-dimensional cube, say a single cell of the original datacube.

In the mathematical notation, slicing is to generate a sub-cube as follows. As an example, let’s start with a 3-dimensional cube $a_{i,j,k}$ with $i \in I, j \in J, k \in K$ and $I = \{i_1, i_2, \dots, i_m\}, J = \{j_1, j_2, \dots, j_n\}$, and $K = \{k_1, k_2, \dots, k_p\}$. Let’s fix the member $k_2 \in K$. The result is a cube of dimension 2, say $b_{i,j}$.

$$b_{i,j} \leftarrow a_{i,j,k_2}$$

means that for each $(i, j) \in I \times J$ tuple in the cube $a_{i,j,k}$, the value of the cell is copied (or mapped) to the corresponding cell in $b_{i,j}$. It is in fact a *matrix-assignment*. Fixing two members in the different dimensions I and K generates a 1-dimensional cube, say c_j :

$$c_j \leftarrow a_{i_3,j,k_2} \quad \text{forall } j \in J$$

The operation \leftarrow means that for each (i, j) -combination in the cube a , the value of the cell is copied to the corresponding cell in b . It is in fact *matrix-assignment*.

7.3.2 Dicing

Dicing (it may be called also *down-sizing*) a n -dimensional datacube means to select a (possibly empty) subset of members from each dimension and discard all tuples which contains at least one member not in the selection (see Figure 7.3).

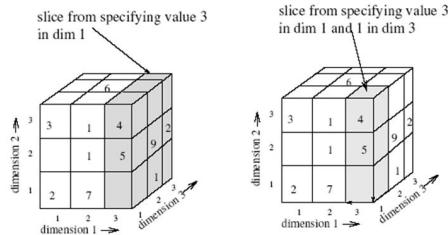


Figure 7.2: Slicing a Cube

Note that the resulting datacube has the same dimension as the datacube before applying the operation. Even if the selection on a particular dimension contains a single member, it is not the same as slicing. The dimension is not reduced and a set containing a single member after all is also a set.

If no member is selected of any dimension, we get the *empty cube*. (Note that the empty cube is not the same than a 0-dimensional cube, which contains a single cell.) If all members are selected from all dimensions, the resulting cube is the same as the original cube.

Mathematically, the operation is simply stated as follows. As above, let's start again with the 3-dimensional cube $a_{i,j,k}$. Let $P_{i,j,k}$ be a predicate (relation) which is true for some (i, j, k) -triples and false for the others. Then dicing $a_{i,j,k}$ on $P_{i,j,k}$ means to create a cube $d_{i,j,k}$ that contains all the (i, j, k) -triples of $a_{i,j,k}$ for which $P_{i,j,k}$ is true. The result is an assignment:

$$d_{i,j,k} \leftarrow a_{i,j,k} \quad \text{forall } (i, j, k) \in P_{i,j,k}$$

A shorter notation is :

$$d_{i,j,k|P} \leftarrow a_{i,j,k}$$

7.3.3 Sizing

Sizing (it may be called also *up-sizing*) a n -dimensional datacube means to add tuples which contains members in a particular dimension that were not part of that dimension. The dimensions are extended: elements are added to one or more dimensions.

The consequence is an extension of the datacube in one or several dimensions. This operation does not change the dimension of the resulting cube – just its size in one or several dimension is changed (see Figure 7.4). A particular operation is when only one dimension is extended. Then this operation can also be seen as “merging” a n -dimensional datacube with a $(n - 1)$ -datacube in which the $n - 1$ dimensions occur in both cubes. It is “adding a slice” to the cube.

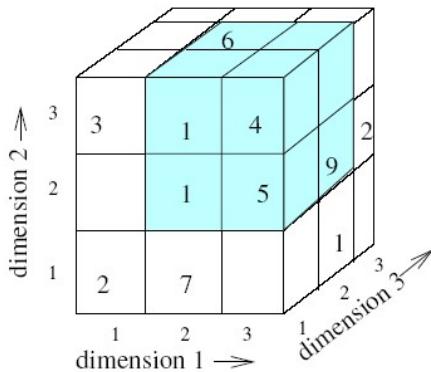


Figure 7.3: Dicing a Cube

This last special case can be implemented as another assignment (where j'_{n+1} is a new element in the dimension J) :

$$f_{i,j,k} \leftarrow \begin{cases} a_{i,j,k} & \text{if } j \in J \\ b_{i,k} & \text{if } j = j'_{n+1} \end{cases}$$

Another important special case occurs when the added $(n - 1)$ -slice is an aggregated cube. A *aggregated cube* is constructed from the n -dimensional cube as follows: Choose one dimension out of the n dimensions in this cube, then choose an operation that can be applied to the cells along the chosen dimension (p.e. summation, maximum, etc., for numerical cells) (see Figure 7.5).

The aggregated cube is an $(n - 1)$ -dimensional cube. Mathematically, it is another assignment:

$$b_{i,k} \leftarrow \sum_{j \in J} a_{i,j,k} \quad \text{forall } i \in I, k \in K$$

Of course, one can also aggregate along the other dimensions i and k , in this example, and one gets the 2-dimensional cubes as follows:

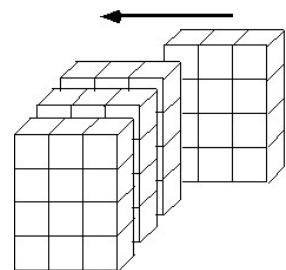


Figure 7.4: Dicing a Cube

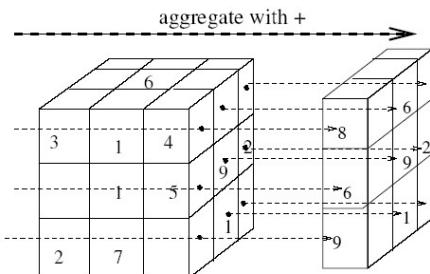


Figure 7.5: A Aggregated Cube (Summation)

$$b'_{i,j} \leftarrow \sum_{k \in K} a_{i,j,k} \quad \text{forall } i \in I, j \in J \quad b''_{j,k} \leftarrow \sum_{i \in I} a_{i,j,k} \quad \text{forall } j \in J, k \in K$$

The aggregation can be continued to generate 1-dimension cubes:

$$c'_i \leftarrow \sum_{j,k} a_{i,j,k} \quad c''_j \leftarrow \sum_{i,k} a_{i,j,k} \quad c'''_k \leftarrow \sum_{i,j} a_{i,j,k}$$

Finally, one gets the 0-dimensional cube by aggregating all cells:

$$d \leftarrow \sum_{i,j,k} a_{i,j,k}$$

Figure 7.6 shows all possible aggregated cubes from a 0 to 4-dimensional cube. Each node in the graph denotes a aggregated cube and the (down)-links represent the aggregation operation. For

example, from the 4-dimensional cube abcd, one can generated a 3-dimensional cube *bcd by aggregation along the dimension a. This shows that the generation of all aggregated cubes forms a lattice.

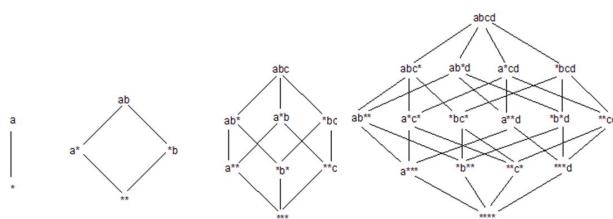


Figure 7.6: Lattice of All Aggregated Cubes

Represented in another way, the 3-dimensional cube of dimension $2 \times 3 \times 4$ (left-bottom cube in Figure 7.7) is augmented by all its aggregated cubes.

In general, from a n -dimensional cube with dimension cardinalities c_1, c_2, \dots, c_n , one can generate $\binom{n}{i}$ aggregated i -dimensional cubes. Totally, there are $2^n - 1$ aggregated cubes. All aggregated cubes together with the original cube can be merged together to form a new cube – the *augmented cube*. It is also an n -dimensional cube, for which all dimensions are extended by just one element.

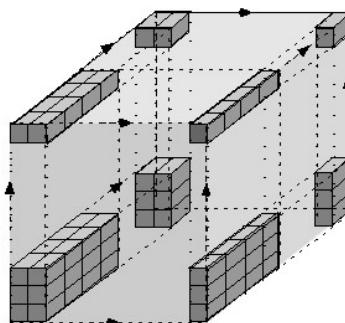


Figure 7.7: All Aggregated Cubes

The order in which the aggregated cubes are build to form an augmented cube is not important supposing the augmented operation is commutative and associative. There are several aggregate operations (for numerical data) that can be considered: Sum, Average, Max, Min, Count, Variance, Standard deviation, and others. More than one aggregate operation may be applied at the same time.

7.3.4 Rising

Rising an n -dimensional cube is an operation that “lifts” the datacube to a higher dimension. This can occur in two important applications. (1) Two

datacubes having the same number of dimensions and having the same dimensions can be merged together. Figure 7.8 displays an example.

This case is especially interesting for comparative studies of two or several datacubes, comparing scenarios and outcome of the same datacube. In LPL this is implemented as *Multiple Snapshot Analysis*. Rising implies to introduce an additional dimension into the resulting datacube. LPL automatically adds a set called `_SNAP_`. Hence the resulting datacube has then $n + 1$ dimensions.

(2) The second important application arises when the actual cube has to be partitioned into several groups, for example, along a time dimension, the months, one has to group the dimension along quarters; or along a product dimension, one has to group them into various product categories, etc (see Figure 7.9).

The original cube is partitioned into the desired parts and build from each part a complete datacube of the original size – by getting eventually a very sparse cube. Then these cubes are risen by “merging” them along a new dimension, which implements the partition, see Figure 7.10.

The idea behind this partition of a cube is that certain dimensions can be structured into hierarchies (Year – Month – Week – Day, Continent – Country – Region – State). The partition can be arbitrarily however, it can be even on several dimensions, that is, certain tuples of the cube may belong to one part and other tuples to another part. The two most important applications of this operation are grouping and hierarchy building in OLAP tool and grouping and subgrouping in reports.

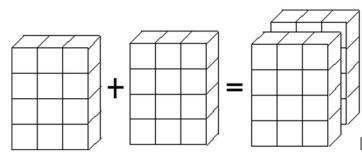


Figure 7.8: Rising Identical Cubes

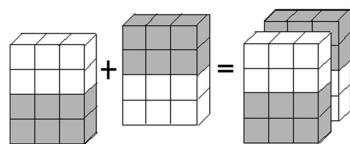


Figure 7.9: Rising Cubes by Extending and Merging Dimensions

7.3.5 Selecting and Ordering

Selecting a subset of elements in one or several dimension results in a datacube of the same dimension with fewer tuples. This operation can be neatly integrated into a visualisation tool of datacube. Another similar operation is ordering: The elements in one or several dimension are sorted along a criterion. For example, a distance matrix can be sorted in the order of the distance (global sorting) to get the 5 shortest distance in the matrix, but one could also

sort it along one dimension (dimensional sorting), to get the 5 closest neighbors of each location. This may be interesting in a heuristic to find a good TSP tour.

7.4. Interpretation of the Operations in the Light of OLAP

On-Line Analytical Processing (OLAP) is a category of software technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user.

OLAP is implemented in a multi-user client/server mode and offers consistently rapid response to queries, regardless of database size and complexity. OLAP helps the user synthesize enterprise information through comparative, personalized viewing, as well as through analysis of historical and projected data in various “what-if” data model scenarios. This is achieved through use of an OLAP Server.

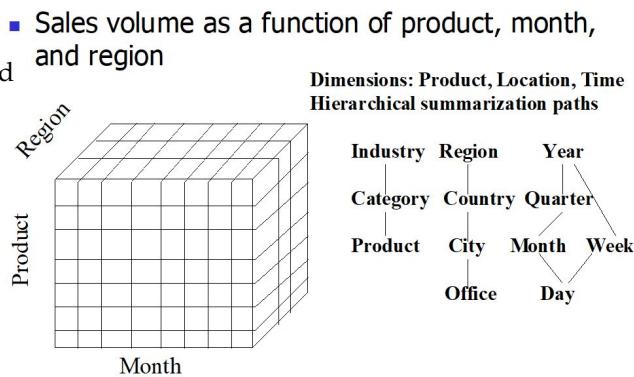


Figure 7.10: Rising Cubes by Extending and Merging Dimensions

We use the OLAP glossary to go through the different “operations” needed to be able to characterize a system an OLAP, see [www.OLAPCouncil.org].

AGGREGATE (CONSOLIDATE, ROLL-UP)

“Multi-dimensional databases generally have hierarchies or formula-based relationships of data within each dimension. Consolidation involves computing all of these data relationships for one or more dimensions, for example, adding up all Departments to get Total Division data. While such relationships are normally summations, any type of computational relationship or formula might be defined.” See Figure 7.11.

The Roll-Up operation is typically to partition a cube and then to rise it a long the partition. This operation can be repeated several times in order to generate

hierarchies of dimensions. Aggregation has been extensively discussed.

DRILL DOWN/UP

"Drilling down or up is a specific analytical technique whereby the user navigates among levels of data ranging from the most summarized (up) to the most detailed (down)."

The drilling paths may be defined by the hierarchies within dimensions or other

relationships that may be dynamic within or between dimensions. For example, when viewing sales data for North America, a drill-down operation in the Region dimension would then display Canada, the eastern United States and the Western United States. A further drill-down on Canada might display Toronto, Vancouver, Montreal, etc."

The Drill-Down means slicing a cube along the hierarchies defined before by a rolling-up process.

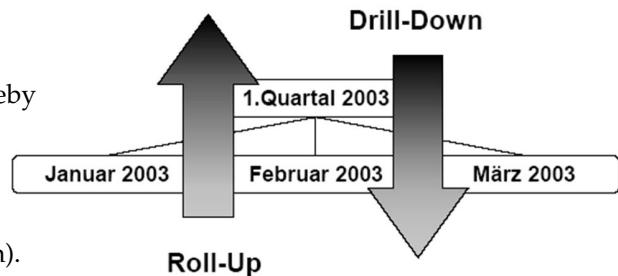


Figure 7.11: Roll-Op/Drill-Down Operations

MULTI-DIMENSIONAL ANALYSIS

"The objective of multi-dimensional analysis is for end users to gain insight into the meaning contained in databases. The multi-dimensional approach to analysis aligns the data content with the analyst's mental model, hence reducing confusion and lowering the incidence of erroneous interpretations. It also eases navigating the database, screening for a particular subset of data, asking for the data in a particular orientation and defining analytical calculations. Furthermore, because the data is physically stored in a multi-dimensional structure, the speed of these operations is many times faster and more consistent than is possible in other database structures. This combination of simplicity and speed is one of the key benefits of multi-dimensional analysis."

Multi-dimensional analysis is nothing else than cube manipulation!

CALCULATED MEMBER

"A calculated member is a member of a dimension whose value is determined from other members' values (e.g., by application of a mathematical or logical operation). Calculated members may be part of the OLAP server database or may have been specified by the user during an interactive session. A calculated member is any member that is not an input member."

By rising a cube, one can add member to a dimensions the cells of which are calculated. The aggregates are typical such calculated cells.

CHILDREN AND HIERARCHICAL RELATIONSHIPS

“Members of a dimension that are included in a calculation to produce a consolidated total for a parent member. Children may themselves be consolidated levels, which requires that they have children. A member may be a child for more than one parent, and a child’s multiple parents may not necessarily be at the same hierarchical level, thereby allowing complex, multiple hierarchical aggregations within any dimension.”

“Any dimension’s members may be organized based on parent-child relationships, typically where a parent member represents the consolidation of the members which are its children. The result is a hierarchy, and the parent-child relationships are hierarchical relationships.”

“Members of a dimension with hierarchies are at the same level if, within their hierarchy, they have the same maximum number of descendants in any single path below. For example, in an Accounts dimension which consists of general ledger accounts, all of the detail accounts are Level 0 members. The accounts one level higher are Level 1, their parents are Level 2, etc. It can happen that a parent has two or more children which are different levels, in which case the parent’s level is defined as one higher than the level of the child with the highest level.”

Using rising a cube it was shown how a hierarchy can be implemented.

NAVIGATION

“Navigation is a term used to describe the processes employed by users to explore a cube interactively by drilling, rotating and screening, usually using a graphical OLAP client connected to an OLAP server.”

See implementation of pivot-tables in LPL.

NESTING (OF MULTI-DIMENSIONAL COLUMNS AND ROWS)

January				February				March			
Actual		Budget		Actual		Budget		Actual		Budget	
Prod A	Prod B	Prod A	Prod B	Prod A	Prod B	Prod A	Prod B	Prod A	Prod B	Prod A	Prod B

Figure 7.12: Horizontal Nesting Display

“Nesting is a display technique used to show the results of a multi-dimensional query that returns a sub-cube, i.e., more than a two-dimensional slice or page.

The column/row labels will display the extra dimensionality of the output by nesting the labels describing the members of each dimension. For example, the display's columns may be as seen in Figure 7.12. These columns contain three dimensions, nested in the user's preferred arrangement."

Chocolate Bars	Unit Sales	xxxx	xxxx	xxxx
	Revenue	xxxx	xxxx	xxxx
	Margin	xxxx	xxxx	xxxx
Fruit Bars	Unit Sales	xxxx	xxxx	xxxx
	Revenue	xxxx	xxxx	xxxx
	Margin	xxxx	xxxx	xxxx

Figure 7.13: Vertical Nesting Display

"Likewise, a report's rows may contain nested dimensions", see Figure 7.13.

This was consistently implemented into the modeling environment of LPL (lplw.exe).

PAGE DISPLAY (PIVOT, ROTATE, ROW DIMENSION, COLUMN DIMENSION, HORIZONTAL DIMENSION, VERTICAL DIMENSION)

"The page display is the current orientation for viewing a multi-dimensional slice. The horizontal dimension(s) run across the display, defining the column dimension(s). The vertical dimension(s) run down the display, defining the contents of the row dimension(s). The page dimension-member selections define which page is currently displayed. A page is much like a spreadsheet, and may in fact have been delivered to a spreadsheet product where each cell can be further modified by the user."

"To change the dimensional orientation of a report or page display we use pivoting. For example, rotating may consist of swapping the rows and columns, or moving one of the row dimensions into the column dimension, or swapping an off-spreadsheet dimension with one of the dimensions in the page display (either to become one of the new rows or columns), etc. A specific example of the first case would be taking a report that has Time across (the columns) and Products down (the rows) and rotating it into a report that has

Product across and Time down. An example of the second case would be to change a report which has Measures and Products down and Time across into a report with Measures down and Time over Products across. An example of the third case would be taking a report that has Time across and Product down and changing it into a report that has Time across and Geography down.”

PAGE DIMENSION

“A page dimension is generally used to describe a dimension which is not one of the two dimensions of the page being displayed, but for which a member has been selected to define the specific page requested for display. All page dimensions must have a specific member chosen in order to define the appropriate page for display.”

See: “take out/in” operator in the next section.

SELECTION

“A selection is a process whereby a criterion is evaluated against the data or members of a dimension in order to restrict the set of data retrieved. Examples of selections include the top ten salespersons by revenue, data from the east region only and all products with margins greater than 20 percent.” See: dicing

SLICE AND DICE

“A slice is a subset of a multi-dimensional array corresponding to a single value for one or more members of the dimensions not in the subset. For example, if the member Actuals is selected from the Scenario dimension, then the sub-cube of all the remaining dimensions is the slice that is specified. The data omitted from this slice would be any data associated with the non-selected members of the Scenario dimension, for example Budget, Variance, Forecast, etc. From an end user perspective, the term slice most often refers to a two-dimensional page selected from the cube.”

“The user-initiated process of navigating by calling for page displays interactively, through the specification of slices via rotations and drill down/up.”

Slicing and dicing have been explained.

7.5. Pivot-Table: 2-dimensional Representation of Datacube

Datacubes must be represented on sheet of papers or on a screen in order to be viewed by human beings, that is, they must be projected onto a two-dimensional space. We call a two-dimensional representation of a cube **pivot-table**.

The figure displays three pivot tables for a 3-dimensional cube $a_{i,j,k}$ with dimensions $i \in \{i1, i2\}$, $j \in \{j1, j2, j3\}$, and $k \in \{k1, k2, k3, k4\}$. The measurement of the cube is $i * j * k$.

- Standard View:** A vertical listing of all tuples in the datacube. The rows are labeled by i and j , and the columns by k . The data values are: $i1j1k1: 1$, $i1j1k2: 2$, $i1j1k3: 3$, $i1j1k4: 4$, $i1j2k1: 2$, $i1j2k2: 4$, $i1j2k3: 6$, $i1j2k4: 8$, $i1j3k1: 3$, $i1j3k2: 6$, $i1j3k3: 9$, $i1j3k4: 12$, $i2j1k1: 2$, $i2j1k2: 4$, $i2j1k3: 6$, $i2j1k4: 8$, $i2j2k1: 4$, $i2j2k2: 8$, $i2j2k3: 12$, $i2j2k4: 16$, $i2j3k1: 6$, $i2j3k2: 12$, $i2j3k3: 18$, $i2j3k4: 24$.
- Horizontal View (Top):** Shows a horizontal view where dimensions i and j are swapped. The rows are labeled by i and j , and the columns by k . The data values are: $i1j1k1: 1$, $i1j1k2: 2$, $i1j1k3: 3$, $i1j1k4: 4$, $i1j2k1: 2$, $i1j2k2: 4$, $i1j2k3: 6$, $i1j2k4: 8$, $i1j3k1: 3$, $i1j3k2: 6$, $i1j3k3: 9$, $i1j3k4: 12$, $i2j1k1: 2$, $i2j1k2: 4$, $i2j1k3: 6$, $i2j1k4: 8$, $i2j2k1: 4$, $i2j2k2: 8$, $i2j2k3: 12$, $i2j2k4: 16$, $i2j3k1: 6$, $i2j3k2: 12$, $i2j3k3: 18$, $i2j3k4: 24$.
- Mixed View (Bottom):** Shows a mixed view where dimension k is swapped with dimension a . The rows are labeled by i and j , and the columns by a and k . The data values are: $i1j1a1k1: 1$, $i1j1a1k2: 2$, $i1j1a1k3: 3$, $i1j1a1k4: 4$, $i1j1a2k1: 2$, $i1j1a2k2: 4$, $i1j1a2k3: 6$, $i1j1a2k4: 8$, $i1j2a1k1: 2$, $i1j2a1k2: 4$, $i1j2a1k3: 6$, $i1j2a1k4: 8$, $i1j2a2k1: 4$, $i1j2a2k2: 8$, $i1j2a2k3: 12$, $i1j2a2k4: 16$, $i1j3a1k1: 3$, $i1j3a1k2: 6$, $i1j3a1k3: 9$, $i1j3a1k4: 12$, $i1j3a2k1: 3$, $i1j3a2k2: 6$, $i1j3a2k3: 9$, $i1j3a2k4: 12$, $i2j1a1k1: 2$, $i2j1a1k2: 4$, $i2j1a1k3: 6$, $i2j1a1k4: 8$, $i2j1a2k1: 2$, $i2j1a2k2: 4$, $i2j1a2k3: 6$, $i2j1a2k4: 8$, $i2j2a1k1: 4$, $i2j2a1k2: 8$, $i2j2a1k3: 12$, $i2j2a1k4: 16$, $i2j2a2k1: 4$, $i2j2a2k2: 8$, $i2j2a2k3: 12$, $i2j2a2k4: 16$, $i2j3a1k1: 6$, $i2j3a1k2: 12$, $i2j3a1k3: 18$, $i2j3a1k4: 24$, $i2j3a2k1: 6$, $i2j3a2k2: 12$, $i2j3a2k3: 18$, $i2j3a2k4: 24$.

Figure 7.14: Pivot Tables by Switching Dimensions Horizontally/Vertically

Datacubes can be represented in many ways on a two-dimensional space. Such representations are shown in Figure 7.14. The standard view – or database view, is a vertical (normally top-down) listing of all tuples in the datacube in a given order. Another is the horizontal view, other are mixtures. The Figure shows an example is the 3-dimensional cube $a_{i,j,k}$ with $i \in \{i1, i2\}$, $j \in \{j1, j2, j3\}$, and $k \in \{k1, k2, k3, k4\}$. The measurement of the cube is $i * j * k$. In general for an n -dimensional cube this kind of switching horizontally/vertically the dimensions gives us $n + 1$ pivot-table representations.

Furthermore, one may consider any permutation order on the dimensions, to obtain $n!$ possibilities of pivot-tables. These permutations are a rich source on projecting the datacube on to a 2-dimensional space. We call this going from one permutation to another *pivoting*. Hence the name of *pivot-table*. Three examples of these permutations are shown in Figure 7.15.

The figure shows three pivot tables derived from a 4-dimensional cube with dimensions i, k, j, a . The first table (left) has dimensions $i, k | j, a$, with data: $i1 k1 | 1 2 3$, $i1 k2 | 2 4 6$, $i1 k3 | 3 6 9$, $i1 k4 | 4 8 12$, $i2 k1 | 2 4 6$, $i2 k2 | 4 8 12$, $i2 k3 | 6 12 18$, $i2 k4 | 8 16 24$. The second table (middle) has dimensions $k | i, j, a$, with data: $k1 | 1 2 2 4 3 6$, $k2 | 2 4 4 8 6 12$, $k3 | 3 6 6 12 9 18$, $k4 | 4 8 8 16 12 24$. The third table (right) has dimensions $k | i, j, a$, with data: $k1 | 1 2 3 2 4 6$, $k2 | 2 4 6 4 8 12$, $k3 | 3 6 9 6 12 18$, $k4 | 4 8 12 8 16 24$.

Figure 7.15: Pivot Tables by Permuting Dimensions

In the left one, 1 dimension is projected horizontally and the permutation is (i, k, j) . The second projects 2 dimensions horizontally and the permutation is (k, j, i) . The right one also projects 2 dimensions horizontally and the permutation is (k, i, j) .

The aggregated cubes can be viewed by “taking out” one or several dimensions from the original cube. So Figure 7.16 represents the aggregated cubes $b_{j,k} = \sum_i a_{i,j,k}$ and $c_{i,k} = \sum_j a_{i,j,k}$. The aggregate operator was SUM.

The figure shows two aggregated cubes. The left cube has dimensions $i | k | b$, with data: $i1 | k1 k2 k3 k4 | 3 6 9 12$, $i2 | 6 12 18 24$, $i3 | 9 18 27 36$. The right cube has dimensions $i | k | c$, with data: $i1 | k1 k2 k3 k4 | 6 12 18 24$, $i2 | 12 24 36 48$.

Figure 7.16: Two Aggregated Cubes

To summarize: Given any n -dimensional cube and a permutation on the dimension as well as two numbers h and k and an aggregate operator, one can generate any pivot-table of the cube or one of its aggregated cubes, h being the number of dimensions that are projected horizontally, and k being the number of dimensions of “taken out”. The permutation is then interpreted as follows: project the first dimensions vertically, the h following horizontally and the last k once are “taken out”. For example, the standard view of a 5-dimensional cube would be given as: $\text{perm} = (1, 2, 3, 4, 5)$, $h = 0$, $k = 0$. The two pivot-tables in Figure 7.16 are defined by:

$$\text{perm} = (2, 3, 1), h = 1, k = 1, \text{op} = \text{sum}$$

and $\text{perm} = (1, 3, 2), h = 1, k = 1, \text{op} = \text{sum}$

Furthermore, *ordering* can be taken into account. Each cell in the datacube is determined by the tuple of its members. Hence, the order in which the tuples are given does not matter. However, one could exploit this freedom to impose a specified order on each dimension's member list. The order than imposed the sequence in which the cells are listed in a particular pivot-table. Any permutation order on the members on each dimension gives a particular pivot-table. The ordering is easy to specify: Given an initial order of the members, one only need to attach a permutation vector to each dimension.

Another operation in showing a cube as a particular pivot-table is *selection*. One can dice a cube first and then display the diced cube as a pivot-table. Another way to view this is to attach a Boolean on each member, and set its value to TRUE if the particular member should be displayed in the pivot-table and FALSE else wise. Hence, each dimension needs a boolean vector of the size of the dimension to specify a selection.

The augmented cube can also be integrated easily into the pivot-table presentations. In the terminology we used earlier, we first *size* the cube with its aggregates and then show the sized cube as a pivot-table. Another way again is to integrate this information into the displaying operations: given a cube, we add (1) a permutation, (2) define a h and k and an (3) an aggregate-op.

Normally however, there is no need to compute all aggregated cubes in order to display them in a particular pivot-table. Let's explain this in the example of a 2-dimensional and then of a 3-dimensional cube.

Figure 7.17 shows a cube of dimension 2 on the left side a pivot-table with $h = 1$. All three aggregated cubes (a^* , $*b$, $**$) are visible and attached as last row and last column. All aggregated cubes are needed (ab , a^* , $*b$, and $**$), see Figure 7.18. The middle pivot-table with $h = 0$ displays also all aggregates. However, one could argue that in the last four rows only the last is needed (the total of all totals), hence, the aggregated cube $*b$ is not needed. A more "natural way" to display the pivot-table would be the picture on the right.

For a 3-dimensional cube with $h=0$ only the aggregates abc , ab^* , a^{**} , $***$ (see Figure 7.18) are needed, with $h = 1$, we need abc , ab^* , a^{**} , $***$, and a^*b , $**c$. The rule is the following: We need all aggregates along maximally two paths from abc to $***$ in the lattice. It is easy to find them. If we have a 5-dimensional cube, for example, with the dimensions $abcde$ (the top node of the lattice) then the dimensions are partitioned into the vertically and horizontally displayed dimensions for a particular pivot-table (say $h = 2$, then we have $abc \mid de$). The two aggregates needed – following the paths in the lattice – are : $ab^* \mid de$ and $abc \mid d^*$. Hence, following the path in which the stars (*) are filled from right

i	j	a
i1	j1	2
	j2	3
	j3	4
	Agg	9
i2	j1	3
	j2	4
	j3	5
	Agg	11
Agg	j1	5
	j2	7
	j3	9
	Agg	20

i	j	a	
i1	j1	2	
	j2	3	
	j3	4	
	Agg	9	
i2	j1	3	
	j2	4	
	j3	5	
	Agg	11	
Agg	j1	5	
	j2	7	
	j3	9	
	Agg	20	

i	j	a
i1	j1	2
	j2	3
	j3	4
	Agg	9
i2	j1	3
	j2	4
	j3	5
	Agg	11
Agg	Agg	20

Figure 7.17: A Pivot Table with Aggregated Cubes

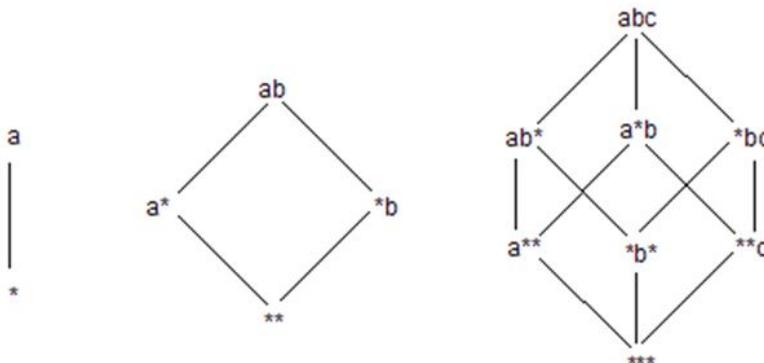


Figure 7.18: A Pivot Table with Aggregated Cubes

to left beginning with the very last entry and beginning with the entry left to 1. Following the path down to **** this way, generates to paths and the corresponding aggregates to calculate and integrate into the pivot-table are given by collecting them in the two paths. Now we also see, why in the cases of $h = 0$ (all vertically) and $h = n - k$ (all horizontally displayed) a single path is needed only. It is easy to see, why this works in general.

Formatting: A pivot-table is – first of all – a 2-dimensional representation of a datacube. The parts of the tables can be thought to be “printed” in rows and columns, in vertically/horizontally arranged cells as shown in Figure 7.19

where the parts are just displayed in the grid without formatting.

Product						
Tiere	Periods	TimeLag	AMOUNT			
		7-9	9-11	11-14	14-18	SUM
Alpha	Per_1	0	3	86	20	27
	Per_2	67	31	16	37	42
	Per_3	8	47	7	84	5
	SUM	29	91	36	77	32
Betas	Per_2	69	84	71	30	16
	Per_3	32	46	24	82	27
	SUM	48	14	87	28	77
SUM	SUM	97	49	88	82	2

Figure 7.19: Unformatted Pivot Table

However, not all cells in the grid have the same meaning. While some “cells” in the pivot-table display the name of the dimensions, others display the data. Basically, a pivot-table can be partitioned into 4 sections: (1) a header, where the dimensions are displayed, (2) the member names of the dimensions to identify a row and a column (3) the aggregates which are “SUM” rows/columns, and (4) the data part. The formatting of these sections is independent from the layout and we are free to enforce visually by colors and other attributes the different sections. An example is shown in Figure 7.20. Different formatting could be chosen.

Product						
Tiere	Periods	TimeLag	AMOUNT			
		7-9	9-11	11-14	14-18	SUM
Alpha	Per_1	0	3	86	20	27
	Per_2	67	31	16	37	42
	Per_3	8	47	7	84	5
	SUM	29	91	36	77	32
Betas	Per_2	69	84	71	30	16
	Per_3	32	46	24	82	27
	SUM	48	14	87	28	77
SUM	SUM	97	49	88	82	2

Figure 7.20: Formatted Pivot Table

Another kind of formatting is data formatting: (1) the data can be formatted

along a mask like `##.###` (with three decimals, if they are numbers), (2) certain data can be shown in a different color (for example if they are negative), (3) The data can be shown as percent of a total, or as difference from a given value, etc.

The spreadsheet software Excel offers the functionality of pivot-tables. But it is a somewhat neglected tool and have serious disadvantages: : (1) The table is a one-way construct, one cannot change any data, (2) certain formats get lost if the table is manipulated by pivoting, (3) it takes quite a time to understand, what you can do, many operations can be done by mans different ways

In the LPL modeling system pivot table manipulation are fully integrated and easy to handle (see user manual of LPL). The LPL modeling system is designed to define and manipulate datacubes. The dimensions must be modeled as SETs and a datacube then is a multi-indexed entity in LPL. For example, to define a 3-dimensional cube one needs the declaration of four entities: 3 SETs, representing the dimensions and a parameter (or a variable, or whatever is indexed).

```
| set i:=[i1 i2]; j:=[j1 j2 j3]; k:=[k1 k2 k3 k4];
| parameter a{i,j,k} := i*j*k;
```

The different operations on datacubes can be implemented in various ways depending often from the context.

Slicing:

```
| parameter b{j,k} := a['i1',j,k];
```

Dicing:

```
| parameter c{i,j,k|a>=10} := a[i,j,k];
```

Sizing:

```
| set h:=[1 i2 i3];
| parameter d{h,j,k|a>=10} := if(h in i, a[h,j,k], sum{i}
| a[h,j,k];
```

Rising:

```
| set i:=[1..3]; j:=[a b c]; k:=[1 2 3 a b c]; m:=[1..2];
| parameter a{i}:=i; b{j}:=10*j;
| c{k}:=if(k<=#i,a[k],b[k-#i]);
| d{m,i}:=if(m=1,a[i],b[i]);
```

The generation of a pivot-table is a function of an extended Write statement. The input information that an algorithm must have to generate a particular pivot-table is:

1. a datacube
2. a permutation of the dimensions , h , k, aggregate operator
3. a order for the members of each dimensions
4. a Boolean for each member indicating of selecting it or not
5. Formatting: (1) of the cell: mask, alignment, font, border, pattern, (2) of the value: as percent, as difference etc., (3) depend on an expression (p.e., negative number with another color).

7.6. Spreadsheets

Spreadsheet modeling represents one of the most successful and widespread applications of present desktop computers. Since their introduction in the late 1970s, spreadsheet programs transformed the way of end-user computing. It created a new paradigm that offers a unique combination of ease of use and unprecedented modeling and calculation power, accessible to every user. As a result, spreadsheet programs became the most widely used decision support tool in modern business. Today's spreadsheet programs are very powerful, versatile, and user friendly. Yet in spite of this technological progress, the basic ideas for building a spreadsheet model remained the same in the last 25 years. Let us briefly explain what these basic ideas are and where they came from. "Spread sheets" have been used by accountants for hundreds of years [12], [58]. The history of accounting and bookkeeping is intrinsically linked to the history of calculation. It is, therefore, not surprising that the first published book on *double-entry-accounting* (Luca Pacioli's famous *Summa de Arithmetica, geometria, proportioni et proportionalita*, at 1494) appeared in a book of mathematics. In the realm of accounting, a "spread sheet" was and is a large sheet of paper with columns and rows that organizes data about transactions for a business person to examine. It spreads all of the costs, income, taxes, and other related data on a single sheet of paper for a manager to examine when making a decision. However, according to [58], until the 19-th century accounting was pure "arithmetics". Only at that time "algebraic" considerations entered the picture in accounting in the form of simple equations such as "Assets = Liabilities + Owner's Equity", compound interest rate calculations or present value considerations. The present value approach in accounting, for instance, played an important role in the German Railway Statutes of 1863 and subsequent legislations.

Spreadsheets are great for small models, although they are used frequently today for complex models with dubious success. The number of papers reporting and analyzing "errors" in spreadsheets is large. As soon as the model becomes complex the disadvantages of using traditional spreadsheets become evident. Since all calculation is cell-oriented, formulas must be copied again and again, revising and rearranging the layout turn out to be a nightmare,

adding a new “item” having the same pattern as the already entered once is error-prune. The logic and structure behind the model get lost or invisible. It became evident already at the beginning of the 1990’s that tools are needed to separate the different concepts: structure and logic, data and presentation. “In many ways, the present state of spreadsheet modeling is reminiscent of the state of data management prior to the ‘database era.’ Before data definition was elevated to the DBMS level, file structures were a fixed part of a program’s code. In a similar vein, the logic and documentation of spreadsheet models are often ‘buried in the formulae,’ and are largely inaccessible to people other than the spreadsheets’ creators. In both cases, the implications are similar: redundant and inconsistent file and model structures, respectively. To complete the analogy, these problems arise because spreadsheet programs lack a high-level means to support the design and maintenance of spreadsheet models.” [34].

In 1991, Lotus – then the leader of spreadsheet technology with its Lotus 1-2-3 – shipped a new product: Improv. Its inventor was Pito Salas, who can be called the father of pivot-table. The concepts of Lotus Improv revolved around the concept of separating the three parts: *data*, *views*, and *formulas*. The data are described in terms of categories such as “Territory”, “Month”, or “Product” and items of those categories such as “Europe”, “January” or “Leather Gloves”. The data can easily be reorganized. Categories are listed on tiles that you simply drag and drop to rearrange your data. Just pick up a category tile and move it to another axis, or rearrange it with respect to the tiles currently on its axis. Because data is associated with categories and items, rearranging the data has no effect on the validity of formulas. Multi-dimensional data – not just 2-dimensional once – can easily be set up. Formulas are written and stored in a separate formula area, which can be viewed simultaneously with the spreadsheet itself. Formulas are written in plain English, making it easier to read and write them. Improv was shipped in 1991 with the new NeXT computer, it was very popular and was one of the killer application on that computer. Unfortunately, Lotus did not port Improv to Windows until 1993. The main reason was that they did not want it to compete directly with their cash-cow: 1-2-3. In the meantime, however, the users already turned away from Lotus 1-2-3 to buy the product Excel. One reason between others was that Lotus put all its effort to the newest version of 1-2-3, which was delayed by more than one year and they did only half-hearted develop Improv. Lotus took Improv from the market. As the inventor of Improv sees it in retro-perspective, is “...that the key strategy mistake was to try to market Improv to the existing spreadsheet market. Instead, if the product were marketed to a segment where the more structured model was a ‘feature’ not a ‘bug’ would have given Lotus the time to learn and improve and refine the model to a point where it would have satisfied the larger market as well.” [Pito Salas at November 28, 2004]. Improv was more a product in the realm of OLAP, a market that just had emerged at that time. It seems “that the OLAP vendors of the time, the likes of Comshare, Holos and IRI were worried about

Improv as it had much the same functionality as product such as Holos and Express but with the backing of Lotus, who at the time were number one in the spreadsheet market with 1-2-3. In the end though, Lotus marketed Improv more as 'spreadsheets done right' (referring to the separation of data and formulas, and the more rigid structure of an Improv model) rather than its OLAP capabilities, which unfortunately had the effect of confusing those customers who now had to choose between 1-2-3 and Improv, and chose Microsoft Excel instead." [Rittman Mark, March 3, 2006]. So, it seems that Improv has failed due to its marketing debacle and less due to its "complexity". It is undeniable that creating a model with Improv requires some preparation and a structured and conceptual approach. However, it is also true that this greatly outweighs the chaotic and "any-thing-goes" of today's spreadsheet building by producing a cleaner and more error-free model.

In any case, the story does not end here. Various attempts have been made to reanimate the ideas of Improv. FlexSheet, for instance, is a free software that implements a subset of Improv. In 2004, a company called Quantrix, has marketed its flagship software Quantrix Modeler, a real superset of Improv and more. Interestingly enough, the inventors of Quantrix Modeler, Peter Murray and Chris Houle, do not see their product as a direct competitor of traditional spreadsheet programs. Quantrix Modeler has even an export function to Excel! However, they analyzed very carefully the deficiencies of the traditional spreadsheet technology [50]:

1. Spreadsheets are two-dimensional: Data that are defined in more than two dimensions are difficult to build. Either one need to spread the data over several work sheets or an arbitrary predefined layout must be specified. Reorganizing the data afterwards is difficult and involving. Excel includes the concept of pivot-table, but they are define only as read-only.
2. Formulas are written with arbitrary coordinates: This make formula cryptic to read and reorganizing the data needs to rewrite many formulas. They are difficult to maintain consistently.
3. Logic is tied to the initial layout of the data: Because the formulas refer directly to cell positions, the logic of the model is inextricably tied to the presentation of the data within the spreadsheet's grid of cells. It is therefore a time-consuming task to rearrange the presentation of the data to highlight or juxtapose particular numbers in the model. Similarly, when trying to present the data through graphs and charts, the user is constrained to work within the program's "wizards", which force the data to be displayed in the way it is depicted within the spreadsheet grid.
4. Lack of dynamism: Modeling is an iterative process. This process must be structured in order to come to a meaningful, consistent and traceable result. Furthermore, the supposed benefits of spreadsheets is the ability

to do "what if" analysis, to work through various scenarios. Changing the content of some cells in order to calculate a variant destroys the "original" model - but these changes are and must be an inevitable part of the spreadsheet modeling process. The task to verify the changes is time-consuming and error-prone. There is no true variant analysis, several variants cannot be inspected at the same time without large reorganization of the work sheets. Users must remember all the formula and cell dependencies and make multiple insertions and deletions to ensure consistency. However, one must remember that the typical spreadsheet developer is a business person not a professional software engineer who is specifically trained in handling such programming details.

5. Replicated logic: In spreadsheet application we often have "similar patterns", since each cell contains its own formula, this means that formulas must be copied and copied again over the whole spreadsheet with their cryptic hardcoded references to other cells. One never can be sure of whether the formulas are copied correctly resulting in momentous and far-reaching errors that are difficult to trace.
6. Row and Column limitations: The most widely used spreadsheet, limits the users to 65,536 rows and 256 columns. When initially designed, this was not considered a problem. However, as models have increased in complexity, this constraint is encountered more frequently.
7. File size: Due to the design of the traditional spreadsheet application that stores a formula in each calculated cell, models quickly become enormous files which are difficult to transfer with email or over a network.

The results are flawed models. This is alarming especially in financial context where a lot of money may be involved.¹ It also results in lack of auditability, in dependencies from its author, in lack of portability and extensibility. One of the main functions of modeling is insight. This is largely absent. Since the logic and structure is mixed with the data and the formatted layout, the structure of the model cannot be separated and inspected on its own. Precisely what is the power of mathematics – insight – is made hidden and intransparent in the spreadsheet, because basically spreadsheets are “arithmetic” not “algebraic”. So many scenarios are not calculated because it is too time-consuming to do it! However, the reality is that business professionals must create increasingly complex models for their decision making processes and their software is based on 25 year old innovations which mapped the paper-ledger into software! There exists an increasing need for better tools.

¹ “A slip of the hand in a computer spreadsheet for bidding on electricity transmission contracts in New York will cost TransAlta Corp. \$24-million (U.S.), wiping out 10 per cent of the company’s profit this year.” ... “A spreadsheet created in 3-months, full-time work by a highly-paid professional has a cost-accounting value in excess of \$25,000.” [?].

Still!

7.7. Conclusion

Data Viewing is an important subject whenever mass of data is involved in modeling. For this we use *reporting*, *data browsing tools* and *data editing tools*. When uniformed data are stored along several dimensions, then we may pack them into datacubes. These data are best viewed and edited through pivot-tables, the 2-dimensional representation of any datacubes.

This paper tries to give a unified theory on datacube and pivoting. It was shown that all data operations in OLAP can by reduced to slicing, dicing, sizing, and rising operations in a multidimensional datacube. The goal is to reduce all kind of proposed operations in OLAP to a few operations in manipulating datacubes. This gives also a new view in implementing OLAP tools. It is, furthermore, important to note that all aspects of data viewing of unified mass of data stored as datacubes can be accessed through pivot-tables. Pivot-tables are easy to understand (if implemented correctly) and easy to manipulate – at least from the point of view of the user. This paper shows that – given a datacube – a few operations and options determine a particular pivot-table. If the user understands these few operations, it is easy to use them. Unfortunately, Excel pivot-tables do not have these properties.

CHAPTER 8

LOGICAL MODELING

“Slow and steady wins the race.”

Logical and Boolean conditions are often used in real-live problem solving. However, its importance in practical modeling contrasts sharply with its treatment in modeling books and courses. Many modeling techniques and applications to formulate logical conditions are exposed in this paper.

It presents various methods of logical modeling, that is, mathematical modeling containing logical propositions and Boolean mixed with mathematical expressions within the constraints. Several modeling techniques are formulated in mathematical notation and in the modeling system LPL (see [72]). Concrete and executable model application examples are given for the different techniques.

A word of caution: It is not trivial in a practical context to use Boolean operators. One needs to analyze carefully the context to use the right operators. “and”, “or”, and other words in a spoken text cannot be translated one-to-one to Boolean “and”, “or”, for instance. Check multiple times before proceeding!

A second word of caution: The modeling formulation may be correct, the implementation of the translation to a mathematical linear model might not always correspond to this logical modeling. This is due to the fact that in most cases, the automatically introduction of a new binary variable is realized in LPL as an implication instead of an equivalence. In most cases, this is justified but not always. Hence, the resulting linear model in LPL should be checked anyway.

A ZIP file of all models can be found [HERE](#)

8.1. Introduction

Mathematical modeling using integer variables is a surprisingly powerful way to formulate and handle real problems. This is not obvious at a first glance. Clearly, if there are objects that cannot be divided into parts, then integer quantities are needed. The number of aeroplanes, for example, must be integer, 1.5 aeroplanes does not make sense in most context. On the other hand, if there are 3'345'678 or 3'345'678.5 hens in a country, the difference may not be important probably, one just can round the number up or down. More important from a modeling point of view, however, are yes/no decisions where integer programming is used, which is the main focus of this paper.

Of course, we know that models containing integer variables are much harder to solve. However, this paper focuses on the various operations which allow the modeler to express the model in a straightforward way and not on how to solve it basically. The focus is on modeling and formulation and less on how to solve these models.

Integer programming can be applied in various contexts:

1. Mathematical problem with logical conditions. This happens often in a mathematical model that must also satisfy some logical conditions. Examples are:
 - a) "If a depot is located at A, then the customer X must be delivered from it."
 - b) "If the journal is too expensive then we must make sure that at least two other journals in the same categories are ordered at a lower price."
 - c) "If product A is manufactured then at least one of product B and C must also be manufactured."
 - d) "If the factory A is closed, then at least one other factory in the region must be kept open."
 - e) "The number of items in a portfolio must not be larger than 5."
 - f) "Two operations A and B cannot be executed at the same time on the machine, either operation A is executed before operation B, or vice versa."
 - g) "Exactly 5 ingredients are allowed in a blend of this type of product."
2. A large category are combinatorial problems. These problems arise when allocating items or persons to different tasks, or carrying out operations in particular order. Examples are:
 - a) Sequencing problems: The sequence of operations must fulfill certain conditions. A particular problem is the *job-shop scheduling* where operations on a machine have to be executed in a defined order. Another particular problem is the *traveling salesman problem*, where the locations must be visited in a sequence that fulfills certain criteria.
 - b) Allocation problems: These problems arise when units or persons have to be allocated to certain tasks, divisions, services or locations, such as the *the depot location problems*.
3. Integer Programming can also be used for modeling non-linear problems. Typical examples are *fixed charge problems*, this occurs when some activity involves a threshold set-up cost, for example. Sometimes it makes sense to remodel a non-linear model as an linear model using additional integer variables. Examples are *piece-wise linear functions* that approximate an otherwise non-linear function.
4. A large category are network problems or problems that arise from graph theory. Some problems are easy problems, such as the *the trans-*

portation problem, the assignment problem, critical path problems, and other flow problems. They all contains a special structure and can be solved without explicitly fix the variables to be integer. Other problems (most of them) are difficult problems, such as *graph vertex coloring, quadratic assignment problem, finding a maximal clique* in a graph and others.

5. Finally, we have problems were one must impose integer values, as already mentioned, such as number of aeroplanes, etc. This is the most evident kind of problems where integer quantities are used, they are also the least important problems from a practical point of view.

Using mathematical notation only, one can formulate all these variants, although one also can use Boolean propositions and predicate logic to express some of these problems. Whether the problem is formulated in a purely mathematical notational framework or partially in logic is basically the same, because one can translate logical expressions into mathematical once, and vice versa. We will see how this can be done in one direction: translating logical expressions into mathematical linear constraints. Advantages of translating logical into mathematical constraints are as follows:

1. Powerful IP- and MIP-solvers can be applied to solve complex problems or at least to find good lower and upper bound for a solution.
2. In the case where the Boolean constraints can be expressed as Horn clauses (Prolog clauses are all Horn-clauses), the model can be solved using LP without branching. Instead of using inference and resolution techniques to solve such problems, one may translate the problem into a LP problem and solve the transformed problem efficiently.
3. Logical and mathematical constraints can be mixed in a single defined modeling language to augment the readability of a model and to produce more compact models.
4. As mentioned, MIP-modeling is extremely rich, but also often tricky and not at all trivial. Often a more natural formulation and representation for many problems is (a subset of) predicate logic.

This paper contains two parts. The first part lists and explains all mathematical and logical operators that can be used in constraints. Model examples are given to understand the semantic of the different operations. The second part presents a translation procedure that translates all logical operators into mathematical linear constraints. This is also basically the procedure that is implemented in the LPL interpreter software. Model examples in LPL are also given.

8.2. The General Model and its Operators

A optimization problem can be formulated as follows:

$$\begin{array}{ll} \min & f(x) \\ \text{subject to} & g_i(x) \leq 0 \quad \text{forall } i \in \{1, \dots, m\} \\ & x \in X \end{array}$$

where f, g_i are functions defined in \mathbb{R}^n (or \mathbb{N}^n), X is a subset of \mathbb{R}^n (or \mathbb{N}^n), and x is a vector of n components x_1, \dots, x_n (see [73]). The notation formalism is given in [67].

Commonly, the functions f, g_i are mathematical functions that consist of the usual operands and operators: *numbers, variables, parameters* and the operators for *addition, subtraction, etc.*, and also function like *sin, log* etc. $f(x)$ is a function that results in a numerical value. The constraints, however, are Boolean expressions: $g_i(x) \leq 0$ that are true or false. Of course, $g_i(x)$ also results in a numerical value which is smaller or equal zero (which means that the constraint holds (is true), otherwise it does not hold (it is false)).

Let us now extend the syntax of the general optimization model: besides the usual operands (numbers, parameters, variables) and numerical operators in a constraint, *Boolean propositions*, and *predicates* as well as Boolean operators can be used. This allows one to mix logical and mathematical notation in one single expressions. In order to make sense for all such expressions, we introduce the convention that the two basic values in logic (*false* and *true*) are mapped to the two numerical values: *zero* and *not-zero* (respectively *one*)¹, this is also common practice in some programming languages. Hence, we adopt the convention for a Boolean proposition x that “ $x = 0$ ” means “ x is false” and in all other cases “ x is true” (in particular when “ $x = 1$ ”). The generalization of a mathematical-logical model is then as follows:

$$\begin{array}{ll} \min & F(x) \\ \text{subject to} & G_i(x) \quad \text{forall } i \in \{1, \dots, m\} \\ & x \in \mathbb{N}^n \text{ (or } \mathbb{R}^n) \end{array}$$

where $F(x)$ and $G_i(x)$ are expressions containing mathematical *and* logical operators. This is a powerful way to compactly specify a large variations of constraints in a uniformed syntax. Before using them in concrete model, these operators and their precedence within an expression is now defined. All the operators and operands that are allowed in a constraint are listed in Table 8.1, x and y are arbitrary (sub)-expressions. The precedence is in decreasing order within the table. Indexed operators are listed in Table 8.2. The indexed operators all have the same precedence.

¹ When using integer numbers there is no ambiguity in the concept of “non-zero”, however if floating point numbers are involved, there must carefully be defined what is meant by “non-zero”: how far away from zero must a number be to be defined as non-zero? A (small) interval must be defined, which results from the context.

Operation	LPL syntax	meaning
$+(-)x$	$+x$	unary plus (minus)
$\neg x$ (\bar{x})	$\sim x$	Boolean negation
(x)	(x)	nesting, changing precedence
x^y	x^y	x power y ($x \geq 0$)
$x \bmod y$	$x \% y$	x modulo y
$\min(x, y)$	$\text{Min}(x, y)$	the smaller of x and y
$\max(x, y)$	$\text{Max}(x, y)$	the larger of x and y
x/y	x/y	division
xy	$x * y$	multiplication
$x - y$	$x - y$	subtraction
$x + y$	$x + y$	addition
$f(x)$	$f(x)$	function like $\sin(x)$ etc.
$x \geq y$	$x \geq y$	greater or equal
$x \leq y$	$x \leq y$	less or equal
$x > y$	$x > y$	greater
$x < y$	$x < y$	less
$x = y$	$x = y$	equal
$x \neq y$	$x \neq y$	not equal
$x \wedge y$	$x \text{ and } y$	Boolean AND
$x \vee y$	$x \text{ or } y$	Boolean OR
$x \dot{\vee} y$	$x \text{ xor } y$	Boolean exclusive-OR
$x \leftrightarrow y$	$x \leftrightarrow y$	Boolean equivalence
$x \rightarrow y$	$x \rightarrow y$	Boolean implication
$x \leftarrow y$	$x \leftarrow y$	Boolean reverse implication
$\overline{x \wedge y}$	$x \text{ nand } y$	Boolean NAND
$\overline{x \vee y}$	$x \text{ nor } y$	Boolean NOR
$x := y$	$x := y$	assignment (returns x)
x, y	x, y	list operator

Table 8.1: Operators and operands in an expression

symbol	LPL syntax	meaning
$\sum_i x_i$	sum{i} x	summation
$\prod_i x_i$	prod{i} x	product
$\min_i x_i$	min{i} x	smallest value
$\max_i x_i$	max{i} x	largest value
$\text{argmin}_i x_i$	argmin{i} x	index of the smallest value
$\text{argmax}_i x_i$	argmax{i} x	index of the largest value
$\text{if}(x, y, z)$	if(x, y, z)	returns y if x is true else z
$\forall_i x_i$	forall{i} x	indexed Boolean AND
$\wedge_i x_i$	and{i} x	indexed Boolean AND
$\exists_i x_i$	exist{i} x	indexed Boolean OR
$\vee_i x_i$	or{i} x	indexed Boolean OR
$\dot{\vee}_i x_i$	xor{i} x	indexed Boolean XOR
$\neg\wedge_i x_i$	nand{i} x	indexed Boolean NAND
$\neg\vee_i x_i$	nor{i} x	indexed Boolean NOR
$\text{atleast}(k)_i x_i$	atleast(k){i} x	at least k must be true
$\text{atmost}(k)_i x_i$	atmost(k){i} x	at most k must be true
$\text{exactly}(k)_i x_i$	exactly{i} x	exactly k must be true

Table 8.2: Indexed Operators in an expression

Three examples that corresponds to a syntactical correct expression are given:

1. “If x and y are true or z is true then and only then w is false”. This is a purely Boolean expression formulated as:

$$x \wedge y \vee z \dot{\vee} w$$

2. “If a is smaller or equal than b or if $3b$ is the same as $4c$ then x or y or both must be true”. This is a mixed expression containing mathematical and logical operators formulated as:

$$(a \leq b \vee 3b = 4c) \rightarrow (x \vee y)$$

3. Let $i \in I$ be a set, P_i be an array of Boolean propositions, and N als be a Boolean proposition that can be true or false. We have the statement: “At most 3 out of all P_i are true or N and exactly one P_i is true, with $i \in I'$ ”. This is an expression with indexed operators formulated as:

$$\text{atmost}(3)_i P_i \vee (N \wedge \dot{\vee}_i P_i)$$

The third expression above may occur in a production planning context. Let P_i be the proposition that “product i is manufactured” and let N be the propo-

sition that “most machines (say at least 80%) are maintained”. Then the constraint may be interpreted as: “At most 3 different products can be manufactured or exactly one product must be manufactured when most machines are maintained (N)”.

The definition of the various Boolean operations is given in Table 8.3.

x	y	$x \wedge y$	$x \vee y$	$x \dot{\vee} y$	$x \rightarrow y$	$x \leftrightarrow y$	$x \leftarrow y$	$x \text{ nand } y$	$x \text{ nor } y$
1	1	1	1	0	1	1	1	0	0
1	0	0	1	1	0	0	1	1	0
0	1	0	1	1	1	0	0	1	0
0	0	0	0	0	1	1	1	1	1

Table 8.3: The Boolean Operators

8.3. Definitions and Logical Identities

The over-bar notation \bar{x} is used equivalently with $\neg x$ for the Boolean negation operator. First some definitions are introduced. A *Boolean proposition* is a statement that can be true or false (1 or 0). For example “It is raining” or “the street is wet” are two statements that can be true or false. Let x and y be two Boolean propositions. A negated or un-negated proposition is called *literal* (for example: x or \bar{y}). An expression formed of literals and an *and* operator is called *conjunction*. An expression formed of literals and an *or* operator is called *disjunction*. A *clause* is a – possibly empty – disjunction of literals. The following two expressions are examples of clauses :

$$\bar{x} \vee \bar{y} \vee z \vee \bar{w} \vee \bar{v}$$

$$x \vee \bar{y} \vee z \vee w \vee \bar{v}$$

A *Horn-clause* is a clause with at most one un-negated literal. Above, the first clause is a Horn-clause while the second is not.² A *conjunctive normal form* (CNF) is a formula which is a conjunction of a set of clauses. A *disjunctive normal form* (DNF) is just a CNF where the two operators \wedge (*and*) and \vee (*or*) are exchanged. In the following formula the first is a CNF, while the second is a DNF :

$$(x \vee \bar{y}) \wedge (y \vee z \vee \bar{w}) \wedge v$$

$$(y \wedge \bar{x}) \vee z \vee (w \wedge \bar{v})$$

² Horn clauses play a basic role in logic programming, such as in the Prolog language.

Various Boolean operators can be transformed into each other (where the symbol \iff is used in the sense “is equivalent to”). 9 identities are listed below.

(1) It is possible to eliminate one of the three operators (*negation, and, or*) using the following identities (Morgan’s law):

$$\begin{array}{lcl} x \wedge y & \iff & \overline{\overline{x} \vee \overline{y}} \\ x \vee y & \iff & \overline{\overline{x} \wedge \overline{y}} \end{array} \quad , \quad \begin{array}{lcl} \overline{x \wedge y} & \iff & \overline{x} \vee \overline{y} \\ \overline{x \vee y} & \iff & \overline{x} \wedge \overline{y} \end{array}$$

(2) The implication and the reverse implication can be replaced using:

$$\begin{array}{lcl} x \rightarrow y & \iff & \overline{x} \vee y \\ x \leftarrow y & \iff & x \vee \overline{y} \end{array}$$

(3) Boolean equivalence can be substituted using:

$$\begin{array}{lcl} x \leftrightarrow y & \iff & (x \rightarrow y) \wedge (x \leftarrow y) \\ (x \vee \overline{y}) \wedge (\overline{x} \vee y) & \iff & (x \wedge y) \vee (\overline{x} \wedge \overline{y}) \end{array} \iff$$

(4) The exclusive *or* is also defined as follows:

$$\begin{array}{lcl} x \dot{\vee} y & \iff & \neg(x \leftrightarrow y) \\ (x \vee y) \wedge (\overline{x} \vee \overline{y}) & \iff & (x \wedge y) \vee (\overline{x} \wedge \overline{y}) \end{array} \iff$$

(5) The operators *nand* and *nor* are given by a negation in Table 8.1:

$$\begin{array}{lcl} x \text{ nand } y & \iff & \overline{x \wedge y} \\ x \text{ nor } y & \iff & \overline{x \vee y} \end{array} \iff \begin{array}{lcl} \overline{x \wedge y} & \iff & \overline{x} \vee \overline{y} \\ \overline{x \vee y} & \iff & \overline{x} \wedge \overline{y} \end{array}$$

(6) The indexed operators have the following meanings (where all x_i are Boolean propositions and $i \in I = \{1, \dots, n\}, n > 0$):

$$\begin{aligned}
 \bigwedge_{i \in I} x_i &\iff (x_1 \wedge x_2 \wedge \dots \wedge x_n) \\
 \bigvee_{i \in I} x_i &\iff (x_1 \vee x_2 \vee \dots \vee x_n) \\
 \bigvee_{i \in I} x_i &\iff \text{exactly}(1)_{i \in I} x_i \quad \iff \sum_{i \in I} x_i = 1 \\
 \text{atleast}(k)_{i \in I} x_i &\iff \sum_{i \in I} x_i \geq k \\
 \text{atmost}(k)_{i \in I} x_i &\iff \sum_{i \in I} x_i \leq k \\
 \text{exactly}(k)_{i \in I} x_i &\iff \sum_{i \in I} x_i = k
 \end{aligned}$$

(7) The *at-least* operator ($\text{atleast}(k)_{i \in I}$) can also be interpreted as a CNF:

$$\text{atleast}(k)_{i \in I} x_i \iff \bigwedge_{S \subset I \mid |S|=n-k+1} \left(\bigvee_{i \in S} x_i \right)$$

The expression on the right hand side is a conjunction of $\binom{n}{n-k+1}$ clauses consisting of exactly k positive propositions each. The *at-least* operator can be interpreted as a compact form of a possibly very large (containing many clauses) CNF.

(8) The four indexed operators *and*, *or*, *nand*, and *nor* can be expressed using the *at-least* operator using :

$$\begin{aligned}
 \bigwedge_{i \in I} x_i &\iff \text{atleast}(n)_{i \in I} x_i && ((1)) \\
 \bigvee_{i \in I} x_i &\iff \text{atleast}(1)_{i \in I} x_i && ((2)) \\
 \text{nand}_{i \in I} x_i &\iff \neg \bigwedge_{i \in I} x_i \iff \bigvee_{i \in I} (\neg x_i) \iff \text{atleast}(1)_{i \in I} (\neg x_i) && ((3)) \\
 \text{nor}_{i \in I} x_i &\iff \neg \bigvee_{i \in I} x_i \iff \bigwedge_{i \in I} (\neg x_i) \iff \text{atleast}(n)_{i \in I} (\neg x_i) && ((4))
 \end{aligned}$$

Interpretation: (1) If and only if all expressions are true then at least all are true; (2) if and only if one expression is true then at least one is true; (3) if not all are true then al least one is false; (4) if none is true then at least all are false.

(9) One may also replace negation of the *at-least*, *at-most*, and *exactly* operators by each other:

$$\begin{aligned}
 \text{atleast}(k)_{i \in I} x_i &\iff \text{atmost}(n - k)_{i \in I} (\neg x_i) \\
 \text{atmost}(k)_{i \in I} x_i &\iff \text{atleast}(n - k)_{i \in I} (\neg x_i) \\
 \text{exactly}(k)_{i \in I} x_i &\iff \text{atleast}(k)_{i \in I} x_i \wedge \text{atmost}(k)_{i \in I} x_i \\
 \neg \text{atleast}(k)_{i \in I} x_i &\iff \text{atmost}(k - 1)_{i \in I} x_i \\
 \neg \text{atmost}(k)_{i \in I} x_i &\iff \text{atleast}(k + 1)_{i \in I} x_i
 \end{aligned}$$

Note also that $\text{atleast}(k)_{...}$ with $k > n$ and $\text{atmost}(k)_{...}$ with $k < 0$ is defined to be false. Likewise, $\text{atmost}(k)_{...}$ with $k \geq n$ and $\text{atleast}(k)_{...}$ with $k \leq 0$ is defined to be true³.

Using the substitutions (1) to (9) given above one can easily see that all logical operators can be reduced to a few once, for example, negation (\neg), and (\wedge), or (\vee), and the *at-least* ($\text{atleast}()$) operator. The next section shows how these identities can be used to translate logical expressions into pure mathematical formulas.

8.4. Translating Boolean Expressions

First of all, it is easy to verify that all Boolean operators in Table 8.3 can also be formulated as mathematical linear expressions if x and y are binary variables with $x, y \in \{0, 1\}$ as shown in Table 8.4. The true (1) and false (0) values of a proposition x in Table 8.3 is interpreted as numerical 1 and 0 values in Table 8.4. The negation $\neg x$ (or \bar{x}) can be substituted by $1 - x$.

x	y	$x \cdot y = 0$	$x + y \leq 1$	$x = 1 - y$	$x \geq y$	$x = y$	$x \leq y$	$x \cdot y = 0$	$x + y = 0$
1	1	1	1	0	1	1	1	0	0
1	0	0	1	1	0	0	1	1	0
0	1	0	1	1	1	0	0	1	0
0	0	0	0	0	1	1	1	1	1

Table 8.4: The Boolean Operators as Mathematical inequalities

Table 8.5 displays a collection of further equivalences and it is easy to check their identity by applying the true/false (0,1) values to each proposition (binary) and compare the result.

However, translating logical constraints into mathematical linear inequalities in a systematical way requires a precise procedure. The general idea is to transform the Boolean expression into a conjunctive normal form eventually

³ In the LPL modeling language syntax, we use the convention that $-n \leq k \leq 0$ means $0 \leq n - k \leq n$.

Propositions: P, X, Y, ...	0-1 binary variables p, x, y, ...
X	$x \geq 1$
$\neg X$ (or \bar{x})	$x \leq 0$
$X \vee Y$	$x + y \geq 1$
$X \vee Y \vee \dots \vee Z$	$x + y + \dots + z \geq 1$
$X \wedge Y$	$x \geq 1, y \geq 1$ (or $x + y \geq 2$)
$X \wedge Y \wedge \dots \wedge Z$	$x \geq 1, y \geq 1, \dots, z \geq 1$ (or: $x + y + \dots + z \geq n$)
$X \rightarrow Y$	$x \leq y$
$X \dot{\vee} Y$	$x + y = 1$
$X \dot{\vee} Y \dot{\vee} \dots \dot{\vee} Z$	$x + y + \dots + z = 1$
$X \leftrightarrow Y$	$x = y$
$X \leftrightarrow Y \leftrightarrow \dots \leftrightarrow Z$	$x = y = \dots = z$
$X \text{ nor } Y$ (or: $\overline{X \vee Y}$)	$x \leq 0, y \leq 0$
$\overline{X \vee Y \vee \dots \vee Z}$	$x + y + \dots + z \leq 0$
$X \text{ nand } Y$ (or: $\overline{X \wedge Y}$)	$x + y \leq 1$
$\overline{X \wedge Y \wedge \dots \wedge Z}$	$x + y + \dots + z \leq n - 1$
$X \leftrightarrow \neg Y$	$x + y \leq 1$
$P \rightarrow X \wedge Y \wedge \dots \wedge Z$	$x \geq p, y \geq p, \dots, z \geq p$ (or: $x + y + \dots + z \geq np$)
$P \rightarrow X \vee Y \vee \dots \vee Z$	$x + y + \dots + z \geq p$
$X \wedge Y \wedge \dots \wedge Z \rightarrow P$	$x + y + \dots + z - p \leq n - 1$
$X \vee Y \vee \dots \vee Z \rightarrow P$	$x \leq p, y \leq p, \dots, z \leq p$ (or: $x + y + \dots + z \leq np$)
$X \wedge (Y \vee Z)$	$x = 1, y + z \geq 1$
$X \vee (Y \wedge Z)$	$x + y \geq 1, x + z \geq 1$ (or: $2x + y + z \geq 2$)
$X_1 \wedge \dots \wedge X_n \rightarrow Y_1 \vee \dots \vee Y_n$	$x_1 + x_2 + \dots + x_n - (y_1 + y_2 + \dots + y_n) \leq n - 1$
$P \leftrightarrow X \wedge Y \wedge \dots \wedge Z$	$x + y + \dots + z - np \leq n - 1,$ $x \geq p, y \geq p, \dots, z \geq p$
$P \leftrightarrow X \vee Y \vee \dots \vee Z$	$x + y + \dots + z \geq p,$ $x \leq p, y \leq p, \dots, z \leq p$

Table 8.5: Logic-mathematical Equivalences

by introducing additional (binary) variables, then it is easy to get the linear inequalities from the resulting clauses using the given identities. Before presenting a systematical procedure, let us try to translate a concrete Boolean expression into purely mathematical linear inequalities (the example is implemented in model [logic1](#)) :

$$((x \wedge y) \vee z) \dot{\vee} w$$

Step 1: replace $a \dot{\vee} b$ with $(a \vee b) \wedge (\bar{a} \vee \bar{b})$. This gives:

$$((x \wedge y) \vee z \vee w) \wedge ((\overline{x \wedge y}) \vee \overline{z} \vee \overline{w})$$

Step 2: Move the negation inwards (using the Morgan's law). This gives:

$$((x \wedge y) \vee z \vee w) \wedge (((\bar{x} \vee \bar{y}) \wedge \bar{z}) \vee \bar{w})$$

Step 3: Move the *or*-operators inwards over the *and*-operators. This gives the following conjunctive normal form (CNF) :

$$(x \vee z \vee w) \wedge (y \vee z \vee w) \wedge (\bar{x} \vee \bar{y} \vee \bar{w}) \wedge (\bar{z} \vee \bar{w})$$

Step 4: Translate the 4 clauses into 4 linear inequalities as follows:

$$\begin{array}{ll} x + z + w & \geq 1 \\ y + z + w & \geq 1 \\ 1 - x + 1 - y + 1 - w & \geq 1 \\ 1 - z + 1 - w & \geq 1 \end{array}$$

8.5. Translating Mixed Expressions

From a practical point of view, mixing logical (Boolean) and mathematical operations in a single formula is much more interesting and its translation into mathematics is more demanding than translating just Boolean formula into mathematics. One way to link Boolean propositions and mathematical (linear) constraints is the big-M method that is presented here⁴.

⁴ Another method to represent a disjunction of linear constraints:

$$A_1x \geq b_1 \vee A_2x \geq b_2 \vee \dots \vee A_nx \geq b_n$$

A disadvantage of the big-M method is that it requires lower and upper bounds on the linear constraints as well as a small number ϵ . In the following, $U(y)$ is used as “upper bound value for y ” and $L(y)$ as “lower bound value for y ”. In the formulas the following shortcuts are used⁵:

$$M = U \left(\sum_i a_i x_i - b \right) , \quad m = L \left(\sum_i a_i x_i - b \right)$$

Case 1: Indicator variable implies an \leq inequality

Rule	Constraint	Translation
50	$\begin{cases} \delta \rightarrow \sum_i a_i x_i \leq b \\ \sum_i a_i x_i > b \rightarrow \bar{\delta} \end{cases}$	$\sum_i a_i x_i - b \leq M \cdot (1 - \delta)$
50'	$\begin{cases} \delta \rightarrow x \leq 0 \\ x > 0 \rightarrow \bar{\delta} \end{cases}$	$x \leq U(x) \cdot (1 - \delta)$
50a	$\begin{cases} \bar{\delta} \rightarrow \sum_i a_i x_i \leq b \\ \sum_i a_i x_i > b \rightarrow \delta \end{cases}$	$\sum_i a_i x_i - b \leq M \cdot \delta$
50a'	$\begin{cases} \bar{\delta} \rightarrow x \leq 0 \\ x > 0 \rightarrow \delta \end{cases}$	$x \leq U(x) \cdot \delta$

Rule 50a is the same as Rule 50, except that δ has been replaced by $\bar{\delta}$. Rule 50' and 50a' are simplified versions of Rule 50 and 50a, the linear inequality has been replaced by just a single variable inequality.

Split the vector x into n components $(x^{(i)}, i \in \{1, \dots, n\})$. Then the disjunction can be formulated as:

$$\begin{aligned} A_i x^{(i)} &\geq b_i \delta_i \\ x = \sum_i x^{(i)} & \\ \sum_i \delta_i &= 1 \\ \delta_i &\in \{0, 1\}, \quad x^{(i)} \geq 0, \quad i \in \{1, \dots, n\} \end{aligned}$$

This formulation has been developed in [59]. The test is also a foundation is the concept of “MIP representability”. An introduction can also be found in [30] and [29].

⁵ LPL automatically calculates and uses these bounds provided that the involved variables are bounded by lower and upper bounds. An error message is generated if this is not the case.

Case 2: Indicator variable is implied by an \leq inequality

Rule	Constraint	Translation
51	$\left. \begin{array}{l} \sum_i a_i x_i \leq b \rightarrow \delta \\ \bar{\delta} \rightarrow \sum_i a_i x_i > b \\ \bar{\delta} \rightarrow \sum_i a_i x_i \geq b + \epsilon \end{array} \right\}$	$\sum_i a_i x_i - b - \epsilon \geq (m - \epsilon) \cdot \delta$
51'	$\left. \begin{array}{l} x \leq 1 \rightarrow \delta \\ \bar{\delta} \rightarrow x > 1 \\ \bar{\delta} \rightarrow x \geq 1 + \epsilon \end{array} \right\}$	$x - 1 - \epsilon \geq (L(x - 1) - \epsilon) \cdot \delta$
51a	$\left. \begin{array}{l} \sum_i a_i x_i - b \leq 0 \rightarrow \bar{\delta} \\ \delta \rightarrow \sum_i a_i x_i > b \\ \delta \rightarrow \sum_i a_i x_i \geq b + \epsilon \end{array} \right\}$	$\sum_i a_i x_i - b - \epsilon \geq (m - \epsilon) \cdot (1 - \delta)$
51a'	$\left. \begin{array}{l} x \leq 1 \rightarrow \bar{\delta} \\ \delta \rightarrow x > 1 \\ \delta \rightarrow x \geq 1 + \epsilon \end{array} \right\}$	$x - 1 - \epsilon \geq (L(x - 1) - \epsilon) \cdot (1 - \delta)$

Case 2 cannot be properly modeled. A small number $\epsilon > 0$ must be introduced to deal with $\sum_i a_i x_i - b > 0$. One may verify the transformation first with the simpler case 51': Suppose the lower bound of x is 0 and $\epsilon = 0.1$, then $(L(x - 1))$ is -1 . Suppose that $x \leq 1$, then the linear inequality reduces to $-1.1 + x \geq -1.1\delta$, and we must derive that $\delta = 1$, suppose now $x \geq 1 + \epsilon$ then the linear inequality reduces again to $s \geq -1.1\delta$ with $s \geq 0$, and δ maybe 0 or 1, verifying the implication.

Case 3: Indicator variable implies an \geq inequality

Rule	Constraint	Translation
52	$\left. \begin{array}{l} \delta \rightarrow \sum_i a_i x_i \geq b \\ \sum_i a_i x_i < b \rightarrow \bar{\delta} \end{array} \right\}$	$\sum_i a_i x_i - b \geq m \cdot (1 - \delta)$
52'	$\left. \begin{array}{l} \delta \rightarrow x \geq 0 \\ x > 0 \rightarrow \bar{\delta} \end{array} \right\}$	$x \geq L(x) \cdot (1 - \delta)$
52a	$\left. \begin{array}{l} \bar{\delta} \rightarrow \sum_i a_i x_i \geq b \\ \sum_i a_i x_i < b \rightarrow \delta \end{array} \right\}$	$\sum_i a_i x_i - b \geq m \cdot \delta$
52a'	$\left. \begin{array}{l} \bar{\delta} \rightarrow x \geq 0 \\ x < 0 \rightarrow \delta \end{array} \right\}$	$x \geq L(x) \cdot \delta$

Rule 52a is the same as Rule 52, except that δ has been replaced by $\bar{\delta}$. Rule 52' and 52a' are simplified versions of Rule 52 and 52a, the linear inequality has been replaced by just a single variable inequality.

Case 4: Indicator variable is implied by an \geq inequality

Rule	Constraint	Translation
53	$\sum_i a_i x_i \geq b \rightarrow \delta$ $\bar{\delta} \rightarrow \sum_i a_i x_i < b$ $\bar{\delta} \rightarrow \sum_i a_i x_i \leq b - \epsilon$	$\sum_i a_i x_i - b + \epsilon \leq (M + \epsilon) \cdot \delta$
53'	$x \geq 1 \rightarrow \delta$ $\bar{\delta} \rightarrow x < 1$ $\bar{\delta} \rightarrow x \leq 1 - \epsilon$	$x - 1 + \epsilon \leq (U(x - 1) + \epsilon) \cdot \delta$
53a	$\sum_i a_i x_i \geq b \rightarrow \bar{\delta}$ $\delta \rightarrow \sum_i a_i x_i < b$ $\delta \rightarrow \sum_i a_i x_i \leq b - \epsilon$	$\sum_i a_i x_i - b + \epsilon \leq (M + \epsilon) \cdot (1 - \delta)$
53a'	$x \geq 1 \rightarrow \bar{\delta}$ $\delta \rightarrow x < 1$ $\delta \rightarrow x \leq 1 - \epsilon$	$x - 1 + \epsilon \leq (U(x - 1) + \epsilon) \cdot (1 - \delta)$

Basically, the four cases (Case 1 to Case 4) can be reduced to two cases in fact. Case 4 is the same as case 1 if we replace $A < B$ by $A \leq B - \epsilon$, or in other words, in the linear inequalities of Case 4 the right hand side b is replaced by $b - \epsilon$ (hence M also changes). Equally, Case 2 can be reduced to Case 3 by replacing $A > B$ by $A \geq B + \epsilon$, or in other words, in the linear inequalities of Case 2 the right hand side b is replaced by $b + \epsilon$ (hence M also changes).

Note also that a case with equalities $\delta \rightarrow \sum_i a_i x_i = b$ can be modeled by applying two rules separately since:

$$\sum_i a_i x_i - b = 0 \iff \left(\sum_i a_i x_i - b \leq 0 \right) \wedge \left(\sum_i a_i x_i - b \geq 0 \right)$$

8.5.1 Some Examples

The five following examples are modeled in [logic0](#) in the modeling language LPL.

Example 1: We would like to decide whether a particular depot should be built or not. The decision to build or not to build can be modeled by a binary

variable δ : $\delta = 1$ mean “to build an depot” and $\delta = 0$ mean’s “not to build the depot”. Now this depends on whether there is a quantity $z > 0$ delivered from this depot. This means: “if there is a certain quantity delivered from a depot then the depot should be built, that is, supposing the upper bound of z is $M > 0$ (see rule 50a):

$$z > 0 \rightarrow \delta = 1 \quad \text{is translated to} \quad z \leq M\delta$$

If equivalence is required, then

$$z > 0 \leftrightarrow \delta = 1 \quad \text{is translated to} \quad \begin{cases} z \leq M\delta \\ z \geq \epsilon\delta; \end{cases}$$

Example 2: Model the following statement: “If a product is manufactured at all then there is an additional setup (or fixed) cost.” Suppose that $z \geq 0$ represents the quantity of a product manufactured at a marginal cost per unit of c_1 . If the product is manufactured at all there is a additional unique setup cost of c_2 . Hence, two cases need to be distinguished:

1. Either nothing is manufactured, then $z = 0$ and total cost = 0.
2. Or something is manufactured, then $z > 0$ and total cost = $c_1z + c_2$

To model this situation, a Boolean (indicator) variable δ is introduced which is true if “something is manufactured” ($z > 0$), if nothing is manufactured the δ is false. Now the indicator variable can be linked with the quantity z by the implication (meaning “if the quantity z is strictly larger than 0 then something is manufactured”):

$$z > 0 \rightarrow \delta \quad \text{or} \quad \bar{\delta} \rightarrow z \leq 0$$

Applying rule 50a gives the linear constraint: $z \leq U(z) \cdot \delta$

The cost function is formulated as: $\text{cost} = c_1z + c_2\delta$

To verify the resulting model, the values of δ are checked:

- (1) if $\delta = 1$ (true) then one has: $z \leq U(z)$, $\text{cost} = c_1z + c_2$
- (2) If $\delta = 0$ (false) then one has: $z \leq 0$, $\text{cost} = 0$

In this case, in LPL it is even not needed to introduce a binary variable. The constraint can be formulated as:

$$c_1z + c_2 \cdot (z > 0)$$

The $z > 0$ is automatically replaced by an additional binary variable.

Example 3: Two quantities must be linked logically. For example: "If product A is included in the blend then also at least a quantity b of product B must be included". That is: $z_A > 0 \rightarrow z_B \geq b$

To model this situation, an indicator variable δ is introduced with the meaning "A is included in the blend" that is: $z_A > 0 \rightarrow \delta$. And the requirement "if product A is included in blend then also at least b of product B must be included" can be modeled as: $\delta \rightarrow z_B \geq b$

The two constraints can be translated to mathematical inequalities by applying the two Rules 51a and 53a, giving:

$$\begin{aligned} z_A &\leq U(z_A) \cdot \delta \\ z_B &\geq L(z_B + b) \cdot \delta \end{aligned}$$

To verify the resulting model, check for the values of δ :

- (1) If $\delta = 1$ (true) then $z_A \leq U(z_A)$, $z_B \geq b$ (supposing $L(z_B) = 0$)
- (2) If $\delta = 0$ (false) then $Z_A \leq 0$, $z_B \geq 0$

Example 4: A constraint A must hold if and only if the indicator variable is true. For example: $2v + 3w \leq 1 \leftrightarrow \delta$. Hence they are:

$$2v + 3w \leq 1 \rightarrow \delta \quad \delta \rightarrow 2v + 3w \leq 1$$

Applying two rules, one gets (supposing $L(v) = L(w) = 0$, $U(v) = U(w) = 1$, $\epsilon = 0.001$):

$$2v + 3w + 4\delta \leq 5 \quad , \quad 2v + 3w + 1.001\delta \geq 1.001$$

Example 5: Model the following constraint, supposing $L(x) = L(y) = 0$, $U(x) = U(y) = 10$, and $\epsilon = 0.0001$:

$$3x + 4y \leq 5 \vee 2x + 3y \geq 4 \leftrightarrow x + 2y \geq 3 \vee 2x + 4y \leq 5$$

The details are given in the text [logic0](#). Note that these examples are from [31] and [30].

8.6. Translation Procedure

The automated translation of a logical constraint to a linear mathematical constraint in LPL is done in several steps applying various rules. Each rule is applied recursively on an (constraint) expression tree. In the following transformation procedure, the index-set $i \in \{1, \dots, n\}$ is used for indexed expressions, the names x and y or their indexed form x_i are used to denote arbitrary

expressions or just variables. The letter x or v_i denotes a single or indexed binary variable.

To understand the transformation procedure, the concept of a “syntax tree” and the concept of “cut-off” are introduced first. A syntax tree is a particular representation of an arbitrary (constraint) expression. Figure 8.1 shows the syntax trees of the following three expressions:

$$3 + 4 \cdot 5 , \quad (3 + 4) \cdot 5 , \quad \neg x \wedge (y \vee z)$$

For binary operators like $+$ (addition) or \vee (or operator), the node (represented as a ellipse) has a left and a right child connected by an arc, for unary operators like \neg there is only a left child (the right is a null pointer, see the node *not*).

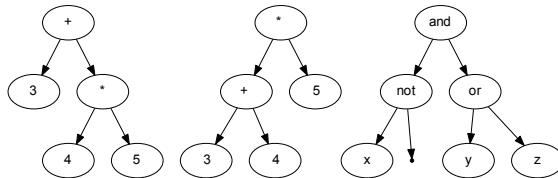


Figure 8.1: Syntax Trees I

Indexed operators as well as function calls are considered as unary operators: See Figure 8.2 which represents the three syntax trees of the expressions:

$$\sum_i x_i y , \quad \bigvee_i (x_i \wedge y) , \quad \text{if}(a < b, x, y)$$

The concept of “cut-off” is the operation of cutting a subtree off the main syntax tree. In mathematics, this operation is called “substitution”: a (sub-)expression is replaced by an additional variable as in the following expression where yz is substituted by an additional variable w :

$$x + yz = 17 \quad \text{becomes} \quad \begin{array}{l} x + w = 17 \\ w = yz \end{array}$$

Represented as a syntax tree, the original tree is decomposed into two (smaller) trees as shown in Figure 8.3, the left tree is decomposed into the two expressions on the right.

Boolean expressions can also be cut-off. In this case, the equality sign ($=$) must be replaced by equivalence (\leftrightarrow). In most cases, however an implication (\rightarrow) is sufficient. An example is given as follows with x and y being Boolean

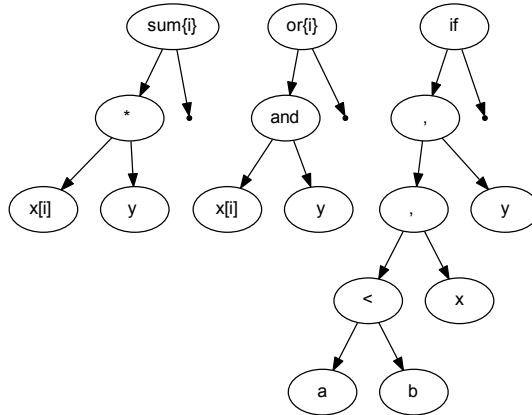


Figure 8.2: Syntax Trees II

propositions and p and q are numerical (real) variables (the trees are shown in Figure 8.4) :

$$(p + q \geq 10) \vee x \wedge y \quad \text{becomes} \quad \delta \vee x \wedge y \\ \delta \rightarrow (p + q \geq 10)$$

In the procedure below, a subtree containing real variables, numerical or relational operators and where the root node is *not* a logical operator, is called *R-subtree*. For example, in the left part of Figure 8.4, the subtree with the root of \geq is a R-tree. The sequence in which the following steps are applied has a large influence on how efficient the resulting expressions are. The sequence can somewhat change depending on the expressions.

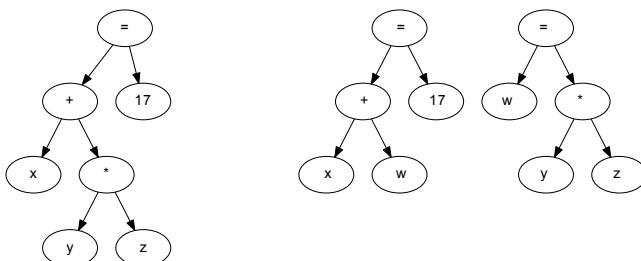


Figure 8.3: Cut-off a Subtree

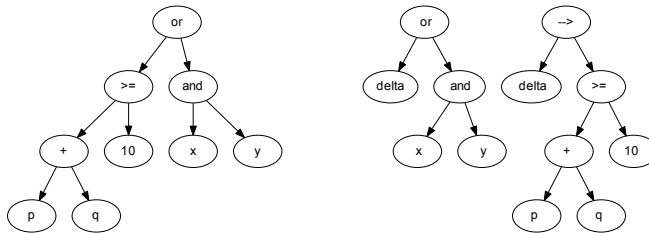


Figure 8.4: Cut-off a Boolean Subtree

Step 0: Cut-off Subtrees

In this step a sub-tree of the syntax-tree is “cut off”. An R-subtree A is cut from a tree T if and only if the path from the root of T to the root of A contains the Boolean operators of equivalence (\leftrightarrow) or the exclusive-or ($\vee\!\vee$).

Step 1: T0-Rules

Several operators are substituted by other operators defined by the previous definitions. Table 8.6 give the list of the operators. Note that all rules are recursively applied to all constraints. (These operators can be eliminated at parse time.)

Rule	Operator	replaced by
01	$x \text{ nor } y$	$\overline{x \vee y}$
02	$x \text{ nand } y$	$\overline{x \wedge y}$
03	$\bigwedge_i x_i$ (indexed-and)	$\forall_i x_i$ (forall-operator)
04	$\bigvee_i x_i$ (indexed-or)	$\exists_i x_i$ (exist-operator)
05	$\bigvee_i x_i$ (indexed xor)	exactly(1) _i x_i (exactly-one)
06	$\text{nor}_i x_i$ (indexed-nor)	atmost(0) _i x_i
07	$\text{nand}_i x_i$ (indexed-nand)	atmost($n - 1$) _i x_i
08	$x \text{ op1 } y \text{ op2 } z$ (op1,op2 $\in \{=, \neq, <, \leq, >, \geq\}$)	$x \text{ op1 } y \wedge y \text{ op2 } z$

Table 8.6: T0-Rules

Step 2: T1-Rules

Several operators are substituted by other operators defined by the previous definitions. Table 8.7 give the list of the operators. Step 1 and Step 2 could be done in a single step.

Rule	Operator	replaced by
10	$x \rightarrow y$	$\bar{x} \vee y$
11	$x \leftarrow y$	$x \vee \bar{y}$
12	$\forall_i x_i$	atleast(n) _i x_i (forall-operator)
13	$\exists_i x_i$	atleast(1) _i x_i (exists-operator)
14	atleast($n - k$) _i x_i	atmost(k) _i \bar{x}_i
15	atmost($n - k$) _i x_i	atleast(k) _i \bar{x}_i
16	exactly(0) _i x_i	atmost(0) _i x_i

Table 8.7: T1-Rules

Step 3: T2-Rules

The NOT-Operator is pushed down the syntax tree and some other reductions are processed. All rules are listed in Table 8.8.

Step 4: Disconnect nested *and*-s

In this step a sub-tree of the syntax-tree is “cut off”. Cut-off means to replace a subtree by an additional binary variable and to add a constraint corresponding to the subtree. Suppose we have the following expression, where E_1 and E_2 are mathematical expressions containing other than binary variables and operators (addition, multiplication, or relational ops) and x and y are two binary variables:

$$x \vee y \vee (E_1 \wedge E_2) \quad [\text{for example: } x \vee y \vee (a \leq 3 \wedge b \geq 2)]$$

In this case, the constraint is replace by the following two constraints (with the additional binary δ):

$$x \vee y \vee \delta \quad , \quad \delta \rightarrow (E_1 \wedge E_2)$$

It is important to note that this cut-off is only applied on a **and** tree node – the and may be indexed – and only if the subtrees contains real variables. A binary **and** node is cut-off only if on the path from the node to the root at least 1 **or** node appears, and a indexed **and** node is only cut-off, if at least 2 **or** nodes appear.

Rule	Operator	replaced by
20	$\neg(\neg x)$	x
21	$\neg(x \wedge y)$	$\neg x \vee \neg y$
22	$\neg(x \vee y)$	$\neg x \wedge \neg y$
23	$\neg(x \leftrightarrow y)$	$x \dot{\vee} y$
24	$\neg(x \dot{\vee} y)$	$x \leftrightarrow y$
25	$\neg(x [\leq, \geq, <, >, =, \neq] y)$	$x [>, <, \geq, \leq, \neq, =] y$
26	$\neg\text{atleast}(k)_i x_i$, ($k \geq 1$)	$\text{atmost}(k-1)_i \neg x_i$
26a	$\neg\text{atleast}(0)_i x_i$	$\text{atleast}(1)_i \neg x_i$
27	$\neg\text{atmost}(k)_i x_i$	$\text{atleast}(k+1)_i \neg x_i$
28	$\neg\text{exactly}(k)_i x_i$, ($k \geq 1$)	$\text{atmost}(k-1)_i x_i \vee \text{atleast}(k+2)_i x_i$
29	$x = 1$ ($1 = x, x \neq 0, 0 \neq x$)	x
29a	$x = 0$ ($0 = x, x \neq 1, 1 \neq x$)	\bar{x}
30	$x = y$	$x \leftrightarrow y$
31	$x \leftrightarrow y$	$(x \vee \bar{y}) \wedge (\bar{x} \vee y)$
32	$x \neq y$	$x < y \vee x > y$
33	$x \dot{\vee} y$	$(x \vee y) \wedge \overline{(x \vee y)}$
34	$\text{exactly}(k)_i x_i$, ($k \geq 1$)	$\sum_i x_i = k$
35	$\text{atmost}(k)_i x_i$, ($k \geq 1$)	$\sum_i x_i \leq k$
35a	$\text{atmost}(0)_i x_i$	$\text{atleast}(0)_i \neg x_i$
36	$\text{atleast}(k)_i x_i$, ($k \geq 2$)	$\sum_i x_i \geq k$

Table 8.8: T2-Rules

Step 5: T3-Rules

In this step remaining **and**-s are pushed upwards in the syntax tree over the **or**-s as much as possible. The rules are listed in Table 8.9. Note that n is the cardinality of the set I and $\text{atleast}(n)_i$ is the same as the indexed **and**. This step can be applied more than once to force a CNF before subtrees are cut.

Rule	Operator	replaced by
40	$x \vee (y \wedge z)$	$(x \vee y) \wedge (x \vee z)$
40a	$(y \wedge z) \vee x$	$(y \vee x) \wedge (z \vee x)$
41	$x \vee \text{atleast}(n)_i y_i$	$\text{atleast}(n)_i (x \vee y_i)$
41a	$\text{atleast}(n)_i y_i \vee x$	$\text{atleast}(n)_i (y_i \vee x)$

Table 8.9: T3-Rules

Step 6: Disconnect Sub-trees

This step is similar to step 4: Sub-trees are “cut off” again. The cut-off is a little bit more involving. The cut-off takes place in several situations

- Like in step 4, remaining *and*-s nested in *or*-s are cut-off in all cases (even if the subtree only contains binary variables) in the same way as in step 4. For example:

$$x \vee (y \wedge z) \quad \text{will become} \quad x \vee \delta, \delta \rightarrow (y \wedge z)$$

- Let E_1 and E_2 again be any mathematical expression. Then any expression of the form

$$E_1 \wedge E_2 \quad \text{will become} \quad E_1, E_2$$

That is, the *and*-operator just decomposes the two operands into two separate constraints.

- To normalize remaining logical expressions, an expression of the form

$$E_1 \vee x \quad \text{will become} \quad x \vee E_1$$

- If a parent node in the syntax tree is not a logical operator and the child node is a logical operator or a relational operator then the subtree is disconnected. Example:

$$a \cdot (b > 0) \quad \text{will become} \quad a \cdot \delta, \delta \leftarrow b > 0$$

We suppose that a is a parameter and b is a variable. (a may also be a variable in this case the model is non-linear.)

5. An expression of the form

$$E_1 \vee E_2 \text{ will become } E_1 \rightarrow \delta, \delta \rightarrow E_2$$

6. Finally, expression with more than one *or*

$$E_1 \vee E_2 \vee x \vee E_3 \text{ will become } \delta_1 + \delta_2 + x + \delta_3 \geq 1, \delta_i \rightarrow E_i (i \in \{1, \dots, 3\})$$

After this step, the only form with logical operators are $x \vee E_1$ or $\bar{x} \vee E_1$.

Step 7: T4-Rules

In this step several modifications are done and the final *or* operator is eliminated.

1. An expression x is transformed to $x \geq 1$, and \bar{x} is transformed into $x \leq 0$, if x is a binary.
2. if \bar{x} occurs in a nested expression then it is transformed to $1 - x$.
3. The two operators $<$ and $>$ are eliminated. If the expressions E_1 and E_2 are integer expressions then $\epsilon = 1$ else $\epsilon = 0.00001$.

$$E_1 < E_2 \text{ will become } E_1 \leq E_2 - \epsilon$$

$$E_1 > E_2 \text{ will become } E_1 \geq E_2 + \epsilon$$

4. Basically, there is only a single logical expression left to be translated: $x \vee E_1$ (or: $\bar{x} \rightarrow E_1$), where E_1 is an equation or inequality.

8.7. Conclusion

A procedure was presented to translate mixed constraints, which contain mathematical and logical operators, into mixed integer constraints – some details have been left out. Such a procedure can be very useful, especially for large mathematical models which are enriched and extended by a few logical constraints. It is also helpful for symbolic manipulation of such constraints. However, there is certainly no claim here that such a translation procedure is practical in all circumstances.

Appendix: Several Model Examples

8.8. Satisfiability I (sat1)

— Run LPL Code , HTML Document –

Logical inference means to deduce logical propositions from a set of premises. The conventional way to solve logical problems is by truth tables, Boolean algebra (transformation laws), and by a well known tree searching procedure, called resolution. Another symbolic method to this inference problem is natural deduction. Still another way is to use numeric (or quantitative) methods to solve an inference problem; that is to convert the Boolean expressions into linear (or non-linear) constraints in order to prove the logical argument using mathematical programming. This has two advantages: (a) in some cases this leads to faster algorithms, (b) the analysis of the quantitative models leads to the unveiling of hidden mathematical structure. Two further advantages are: (c) symbolic (logical) and numeric (mathematical) knowledge can be mixed and represented in the same framework, (d) default or other non-monotonic knowledge can also be modeled using the same framework.

The *satisfiability problem (SAT)* is to decide whether a Boolean statement F follows from another statement E ; or, in other words, whether $E \rightarrow F$ is valid. This problem was the first problem proven to be NP-complete. The model here is a small instance of the general satisfiability problem.

One way to solve this problem using mathematical programming is to translate E and F into two CNF. From these two CNFs it is easy to build the two linear system $\mathbf{Ax} \geq \mathbf{a}$ and $\mathbf{Bx} \geq \mathbf{b}$. Then we solve the problem:

$$\begin{array}{ll} \min & x_0 \\ \text{subject to} & x_0 e + \mathbf{Bx} \geq \mathbf{b} \\ & \mathbf{Ax} \geq \mathbf{a} \\ & x \in \{0, 1\} \end{array}$$

(where e is a column unit vector, x_0 is an additional binary variable). If the minimum value x_0 is 1, then $E \rightarrow F$ is valid, that is, E implies F .

Another method, that does not use an additional binary x_0 , is to pick an arbitrary inequations (clause) within $Bx \geq b$, say $B^{(k)}x \geq b_k$, and to solve the linear problem

$$\begin{array}{ll} \min & B^{(k)}x \\ \text{subject to} & B^{(i)}x \geq b_i \quad \forall i : i \neq k \\ & \mathbf{Ax} \geq \mathbf{a} \\ & x \in \{0, 1\} \end{array}$$

If $[B^{(k)}x^*] \geq b_k$ (with the optimal solution x^*), then E implies F . (LPL adopt the second method).⁶ Here is a small problem instance.

⁶Interestingly, the expression $B^{(k)}x^*$ needs not to be integer. Hooker [37] asserts that

Problem: Is the following argument correct? "If fallout shelters are built, other countries will feel endangered and our people will get a false sense of security. If other countries will feel endangered they may start a preventive war. If our people will get a false sense of security, they will put less effort into preserving peace. If fallout shelters are not built, we run the risk of tremendous losses in the event of war. Hence, either other countries may start a preventive war and our people will put less effort into preserving peace, or we run the risk of tremendous losses in the event of war." (see [28]).

Modeling Steps:

1. The following 6 Boolean propositions (binary variables) are introduced:

p "fallout shelters are built"

q "other countries will feel endangered"

r "other people will get a false sense of security"

s "other countries may start a preventive war"

t "our people will put less effort into preserving peace"

u "we run the risk of tremendous losses in the event of war"

1. Then the first statement can be formulated as: $p \rightarrow q \wedge r$.

2. The second is to be formulated as: $q \rightarrow s$.

3. The third is: $r \rightarrow t$.

4. The fourth is: $\neg p \rightarrow u$.

5. And the final last statement is: $s \wedge t \vee u$.

We therefore have to prove $E \rightarrow F$, with :

$$E \iff (p \rightarrow q \wedge r) \wedge (q \rightarrow s) \wedge (r \rightarrow t) \wedge (\neg p \rightarrow u) \quad \text{and} \quad F \iff (s \wedge t \vee u)$$

Transforming the expressions into CNFs gives:

$$E \iff (\bar{p} \vee q) \wedge (\bar{p} \vee r) \wedge (\bar{q} \vee s) \wedge (\bar{r} \vee t) \wedge (p \vee u)$$

$$F \iff (u \vee s) \wedge (u \vee t)$$

88% of all random generated SAT problems can be solved this way without branching.

We choose the first clause in F :

$$\min B^{(k)}x \iff \min u + s$$

The other clause in F is $B^{(i)}x \geq b_i \iff u + t \geq 1$

The 5 clauses in E corresponds to the five inequalities as follows:

$$-p + q \geq 0, \quad -p + r \geq 0, \quad -q + s \geq 0, \quad -r + t \geq 0, \quad p + u \geq 1$$

The complete optimization problem is:

$$\begin{aligned} & \min && u + s \\ & \text{subject to} && u + t \geq 1 \\ & && -p + q \geq 0 \\ & && -p + r \geq 0 \\ & && -q + s \geq 0 \\ & && -r + t \geq 0 \\ & && p + u \geq 1 \\ & && p, q, r, s, t, u \in \{0, 1\} \end{aligned}$$

Listing 8.1: The Complete Model implemented in LPL [72]

```

model SAT1 "Satisfiability I";
binary variable
  p "fallout shelters are built";
  q "other countries will feel endangered";
  r "other people will get a false sense of security";
  s "other countries may start a preventive war";
  t "our people will put less effort into preserving
     peace";
  u "we run the risk of tremendous losses in the event
     of war";
constraint
  s1: p -> q and r;
  s2: q -> s;
  s3: r -> t;
  s4: ~p -> u;
minimize
  s5: s and t or u;
Write('%s', if (s5,'The argumentation is correct','The
           argumentation is not correct'));
--Write('EQU');
end

```

The IP solver finds that $u + s \geq 1$ – that is the objective function is larger or equal than 1. This means that the argument is correct.

Further Comments: The common technique to solve this problem is by *resolution and unification*, a technique well known in logic and in logic programming languages, such as Prolog. The first step is to negate the consequence and to transform the whole statement into a conjunctive (or disjunctive) normal form, then to prove a contradiction. The CNF of $E \rightarrow \neg F$ of the problem is as follows:

$$\begin{array}{ll} \bar{p} \vee q & (1a) \\ \bar{p} \vee r & (1b) \\ \bar{q} \vee s & (2) \\ \bar{r} \vee t & (3) \\ p \vee u & (4) \\ \bar{s} \vee \bar{u} & (5a) \\ \bar{t} \vee \bar{u} & (5b) \end{array}$$

A possible resolution sequence is:

- (4) together with (5a) generates the resolvent: $p \vee \bar{s}$ (6),
- (1a) together with (2) generates the resolvent: $\bar{p} \vee s$ (7),
- (6) together with (7) generates the resolvent: $\bar{p} \vee p$.

which produces an empty clause, proving the contradiction. Hence the argument is valid.

Questions

1. Verify how LPL translates these formulae into 0-1-constraints by generating the EQU-file. (Add the Instruction `Write('EQU');`).
2. Form the LP relaxation of the linear problem generated by LPL and solve it. What do you observe?

Answers

1. The result of LPL is the same as derived above :

```
min s5: + u + s
s1: + q - p >= 0
s2: + s - q >= 0
s3: + t - r >= 0
s4: + u + p >= 1
X37: + r - p >= 0
X38: + u + t >= 1
```

2. Solve the following LP relaxation

<pre>model x; variable p;q;r;s;t;u;</pre>

```
minimize obj: + u + s;
constraint
    s1: + q - p >= 0;
    s2: + s - q >= 0;
    s3: + t - r >= 0;
    s4: + u + p >= 1;
    X58X: + r - p >= 0;
    X59X: + u + t >= 1;
end
```

The optimum is 1. Since we have $\lceil B^{(k)}x^* \rceil \geq 1$ and $b_k = 1$, we also have $\lceil B^{(k)}x^* \rceil \geq b_k$, and the argument has been proven using the LP relaxation only. By the way, all clauses are *Horn clauses*, therefore, the problem is solvable by the LP-relaxation.

8.9. Satisfiability II (sat2)

— Run LPL Code , HTML Document —

Problem: Is the following set of statements logically consistent? “If the bond market goes up or if interest rates decrease, either the stock market goes down or taxes are not raised. The stock market goes down when and only when the bond market goes up and taxes are raised. If interest rates decrease, then the stock market does not go down and the bond market does not go up. Either taxes are raised or the stock market goes down and interest rates decrease.” (see [28]).

Modeling Steps:

1. We introduce four Boolean propositions:

```
p "the bond market goes up"
q "interest rates decrease"
r "the stock market goes down"
s "taxes are not raised"
```

2. Then the first statement can be formulated as: $(p \vee q) \rightarrow (r \vee s)$.
3. The second then is: $r \leftrightarrow (p \wedge \bar{s})$.
4. The third is: $q \rightarrow (\bar{r} \wedge \bar{p})$.
5. The final statement (the consequence) is: $\bar{s} \vee (r \wedge q)$

Hence we need to prove the following statement:

$$[((p \vee q) \rightarrow (r \vee s)) \wedge (r \leftrightarrow (p \wedge \bar{s})) \wedge (q \rightarrow (\bar{r} \wedge \bar{p}))] \rightarrow [\bar{s} \vee (r \wedge q)]$$

We minimize the last statement $\bar{s} \vee (r \wedge q)$. If the minimal value of this expression is 1, then we know, that the argument holds. Why? Because this expression must be true under all interpretations. Therefore it follows from the first three statements.

Listing 8.2: The Complete Model implemented in LPL [72]

```
model SAT2 "Satisfiability II";
binary variable
  p "the bond market goes up";
  q "interest rates decrease";
  r "the stock market goes down";
  s "taxes are not raised";
constraint
  s1: (p or q) -> (r or s);
```

```

s2: r <-> (p and ~s);
s3: q -> (~r and ~p);
minimize s4: ~s or (r and q);
Write('%s', if (s4,'The argument is consistent','The
argument is not consistent'));
end

```

LPL translates this statement into the following linear model:

```

min: + r - s;
s1: + s + r - p >= 0;
s2: + s - p + r >= 0;
s3: - r - q >= -1;
X55X: + s + r - q >= 0;
X56X: + p - r >= 0;
X57X: - p - q >= -1;
X58X: + q - s >= 0;
X59X: - s - r >= -1;

```

Solution: The argument is *not* correct.

Further Comments: Note: Williams [28] gives the following formula (22): $p + q + r \leq 1$ for s_3 which is not correct (a correct formula is produced by LPL) also formula (20) in Williams is not correct.

Questions

1. Verify that LPL's translation is correct, by generating the CNF using the method given in model **sat1**
2. Verify that the argument *cannot* be proven with the LP relaxation.

Answers

1. Go to model **sat1**
2. Build the LP relaxation of the problem and solve it. The optimum is -1 . This is smaller than the value of $B(k)$ which is 0 . Therefore we have no proof.

8.10. Satisfiability III (**sat3**)

— Run LPL Code , HTML Document –

Problem: Is the following set of statements correct? “If Superman were able and willing to prevent evil, he would do so. If Superman were unable to prevent evil, he would be impotent and if he were unwilling to prevent evil, he would be malevolent. Superman does not prevent evil. If Superman exists, he is neither impotent nor malevolent. Therefore, Superman does not exist.” (see [24]).

Modeling Steps:

1. We introduce 6 Boolean propositions as follows:

```
a "Superman is able to prevent evil"
w "Superman is willing to prevent evil"
i "Superman is impotent"
m "Superman is malevolent"
p "Superman prevents evil"
e "Superman exists"
```

2. Then the first statement can be formulated as: $a \wedge w \rightarrow p$.
3. The second is: $(\bar{a} \rightarrow i) \wedge (\bar{w} \rightarrow m)$.
4. The third is: \bar{p} .
5. The fourth is: $e \rightarrow i \vee m$.
6. The last statement (the consequence) is: \bar{e} .

Hence, we need to show the following Boolean expression:

$$[(a \wedge w \rightarrow p) \wedge (\bar{a} \rightarrow i) \wedge (\bar{w} \rightarrow m) \wedge \bar{p} \wedge (e \rightarrow i \vee m)] \rightarrow (\bar{e})$$

Listing 8.3: The Complete Model implemented in LPL [72]

```
model SAT3 "Satisfiability III";
binary variable
  a "Superman is able to prevent evil";
  w "Superman is willing to prevent evil";
  i "Superman is impotent";
  m "Superman is malevolent";
  p "Superman prevents evil";
  e "Superman exists";
constraint
  s1: a and w -> p;
  s2: (~a -> i) and (~w -> m);
```

```

s3: ~p;
s4: e -> i nor m;
minimize ne: ~e "Superman does not exist";
Write('%s', if (e,'Superman exists.','Superman does not
exist'));
end

```

Solution: The Boolean statement is correct, that is, the argumentation is correct and unfortunately, Superman does not exist.

Questions

1. Verify that LPL's translation is correct, by generating the CNF using the method given in model **sat1**
2. Verify that the argument *cannot* be proven with the LP relaxation.

Answers

1. The LPL translation is as follows

```

min: -e;
s1: p - w - a >= -1;
s2: i + a >= 1;
s4: -i - e >= -1;
X56X: m + w >= 1;
X57X: -m - e >= -1;

```

2. Build the LP relaxation of the problem and solve it. The optimum is -1 . This is smaller than the value of $B(k)$ which is 0 . Therefore we have no proof.

8.11. Satisfiability IV (**sat4**)

— Run LPL Code , HTML Document –

Problem: Check whether x_2 follows from the following conjunctive normal form (CNF):

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_3})$$

Modeling Steps:

1. Each of the five clauses can be formulated as a constraint, since a list of constraints is the same as a conjunction of constraints.
2. Minimize the expression x_2 subject to the five clauses is all we need to do. Why? Suppose minimizing x_2 result in $x_2 = 1$. This means that it is impossible to set $x_2 = 0$. We conclude that if the five clauses are true then x_2 must also be true. Suppose conversely, that the minimizing gives $x_2 = 0$. This means that x_2 does not follow from the five clauses, since the five constraints are true and x_2 is false. Hence x_2 does not follow from the constraints.
3. We conclude that the objective function expression follows from the constraints if and only if its value is greater or equal one (1).

Listing 8.4: The Complete Model implemented in LPL [72]

```

model SAT4 "Satisfiability IV";
binary variable x1; x2; x3; x4;
constraint
  R1: x1 or x2 or x3;
  R2: ~x1 or x2 or ~x4;
  R3: ~x1 or x3;
  R4: ~x1 or ~x3 or x4;
  R5: x1 or ~x3;
minimize obj: x2;
Writep(x1,x2,x3,x4);
end

```

Solution: Since the optimal value for this model is $x_2 = 1$, we conclude that x_2 follows from the five clauses.

Further Comments: LPL translates this Boolean model into a pure 0-1 linear program as follows:

```

model SAT4;
binary variable x1; x2; x3; x4;

```

```
constraint
R1: x1+x2+x3  >= 1;
R2: -x1+x2-x4 >= 1-2;
R3: -x1+x3    >= 1-1;
R4: -x1-x3+x4 >= 1-2;
R5: x1-x3    >= 1-1;
minimize obj: x2;
end
```

8.12. Satisfiability V (Horn Clauses) (**sat5**)

— Run LPL Code , HTML Document —

Problem: Check whether x is implied by the three following (Horn) clauses:

$$x \wedge y \rightarrow z, \quad w \wedge z \wedge y \rightarrow x, \quad x \rightarrow y$$

Modeling Steps: It is a CNF (Conjunctive Normal Form) where the clauses are Horn clauses (clauses that have at most one positive literal, see **sat1**).

Listing 8.5: The Complete Model implemented in LPL [72]

```
model SAT5 "Satisfiability V (Horn Clauses)";
  binary variable x; y; z; w;
  constraint
    R1: x and y -> z;
    R2: w and z and y -> x;
    R3: x -> y;
  minimize obj: x;
  Writep(x,y,z,w,obj);
end
```

If a model consists of Horn clauses then one can solve the LP relaxation of the corresponding 0-1-ILP to solve the problem, although the solution may not be necessarily have an integer solution. We can nevertheless deduce the result from the optimal value. Hence, no branching is needed.

Solution: In this model, the LP relaxation give an optimal value of zero. Hence, x does not follow from the premises. As one can easily verify, the feasible points are:

```
(x,y,z,w) =
(1,1,1,1)
(1,1,1,0)
(0,1,1,0)
(0,1,0,1)
(0,1,0,0)
(0,0,1,1)
(0,0,1,0)
(0,0,0,1)
(0,0,0,0)
```

This list shows that x is not true under all interpretations.

Further Comments: This model corresponds to the following Prolog program (since all constraints are Horn clauses):

```

z :- x, y           // three rules
x :- w, z, y
y :- x
?- x               // the goal to prove

```

Questions

1. Translate the following Prolog program into LPL and solve it. What is the optimum of the 0-1-ILP, what is the optimum of the LP relaxation?

```

x, y, z
x :- y, z
?- y, z

```

Answers

1. The model in LPL is as follows and gives the optimal value solution of -1

```

model Horn1;
binary variable x; y; z;
constraint
  c1: ~x or ~y or ~z;
  c2: x or ~y or ~z;
minimize obj: ~y or ~z;
end

```

The LP relaxation is as follows and give an optimum of -1.5:

```

model Horn1;
variable x[0..1];y[0..1];z[0..1];
minimize obj: -z-y;
constraint c1: -z - y - x >= -2;
                  c2: -z - y + x >= -1;
end

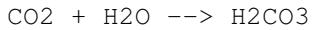
```

(The expression $\sim y$ or $\sim z$ is a separating cut.)

8.13. Chemical Synthesis (sat6)

— Run LPL Code , HTML Document —

Problem: The following chemical reactions are given:



Show that from a blend of MgO , H_2 , O_2 , and C , the molecule H_2CO_3 will be produced.

Modeling Steps: To be able to model this with Boolean logic, some suppositions and simplifications are needed:

1. A Boolean variable for each molecule is introduced with the meaning that the molecule “takes part or does not take part” of a chemical reaction.
2. The molecules which are the inputs in a reaction are linked by a \wedge -connector meaning that they must all be present in a chemical reaction at the same time.
3. Also the molecules which are output of a chemical reaction are connected by a \wedge operator meaning that they are all produced at the same time.
4. Input and outputs of a reactions are linked by implication. This expresses the fact that if the inputs are present then the reaction takes place and the outputs must result.

With these assumptions, the “reaction” is modeled as a simple Boolean (satisfiability) problem. It must be shown that:

$$E \rightarrow F$$

$$\text{with } E = (\text{MgO} \wedge \text{H}_2 \rightarrow \text{Mg} \wedge \text{H}_2\text{O}) \wedge (\text{C} \wedge \text{O}_2 \rightarrow \text{CO}_2) \wedge (\text{CO}_2 \wedge \text{H}_2\text{O} \rightarrow \text{H}_2\text{CO}_3) \wedge \text{MgO} \wedge \text{H}_2 \wedge \text{O}_2 \wedge \text{C}$$

$$F = \text{H}_2\text{CO}_3$$

Listing 8.6: The Complete Model implemented in LPL [72]

```
model Make_H2CO3 "Chemical Synthesis";
binary variable MgO; H2; O2; C; H2CO3; Mg; CO2; H2O;
constraint
  A1: MgO and H2 -> Mg and H2O;
  A2: C and O2 -> CO2;
```

```
A3: CO2 and H2O -> H2CO3;
A4: MgO and H2 and O2 and C;
minimize obj: H2CO3;
Write('Yes, H2CO3 is produced');
Write('No, H2CO3 is not produced');
end
```

The constraints A1 to A3 form the reaction. A4 means that these four molecules are present. In the implementation, E is the set of constraints (which must hold). The objective function represents F. It is minimized, if the objective function F is zero (false) then F does not follow from E, that is $E \rightarrow F$ is false. On the other side, if F is one (true), then F is the consequence of E and $E \rightarrow F$ is true.

Solution: In this model, H_2CO_3 can be (logically) produced from the reaction.

8.14. Simple expressions (logic0)

— Run LPL Code , HTML Document —

Problem: Study how the five following modeling situation can be handled and translated into a mathematical 0-1-ILP formulation (these examples are all from [31] and [30]).

1. We would like to decide whether a particular depot should be built or not. The depot should be built, if a product is delivered to at least one costumer from that depot.
2. If a product is manufactured at all then there is an additional setup (or fixed) cost.
3. If product A is included in the blend then also at least a quantity 4 of product B must be included.
4. If and only if we open a mine then a certain linear condition must hold.
5. Formulate the condition matheamtically: if and only if $3x + 4y \leq 5 \vee 2x + 3y \geq 4$ holds then also $x + 2y \geq 3 \vee 2x + 4y \leq 5$ must hold.

Modeling Steps: [Run the model and generate the EQU-file to see how LPL translates the logical statements into Math.]

Listing 8.7: The Complete Model implemented in LPL [72]

```
model Logic0 "Simple expressions";
parameter c:=5 "choose example";
-- Example 1
variable z [0..10]; binary d;
constraint C1 if c=1: z>0 -> d=1;
constraint C1a if c=1: z>0 <-> d=1;
--Example 2
variable cost;
constraint C2 if c=2: cost = 5*z + 8*(z>0);
-- Example 3
variable zA [0..20]; zB [0..30];
constraint C3 if c=3: zA>0 -> zB>=4;
constraint C3a if c=3: zA=4 -> zB=5;
-- Example 4
variable v [0..1]; w [0..1];
constraint C4 if c=4: 2*v+3*w <= 1 <-> d;
set i:=1..5;
variable aa{i};
constraint C4a if c=4: d or and{i} (aa>5);
-- Example 5
variable x[0..4]; y [0..5];
```

```

constraint C5 if c=5: 3*x+4*y<=5 or 2*x+3*y>=4 <-> x+2*y>=3 or 2*x+4*y<=5;
-- Example 6
binary variable x1; x2; x3;
constraint D1 if c=6: x3 = (x1 or x2);
constraint D2 if c=6: x3 = (x1 and x2);
constraint D3 if c=6: x1 = ~x2;
solve 'nosolve';
Write('EQU');
end

```

Example 1: The decision to build or not can be modeled by a binary variable δ : $\delta = 1$ mean “to build an depot” and $\delta = 0$ mean’s “not to build the depot”. Now this depends on whether there is a positive quantity $z > 0$ delivered from this depot. Hence we may have: “if there is a certain (not-zero) quantity delivered from the depot then the depot should be built”, that is (let M be an upper bound for z):

$$z > 0 \rightarrow \delta = 1$$

This can be modeled by the following linear inequality:

$$z \leq M\delta$$

Note that M is positive number, so $M\delta$ can only be strictly positive if $\delta = 1$, therefore, if $x > 0$ then δ must be 1). Inversely, if the depot is not built then nothing can be delivered from that depot ($\delta = 0 \rightarrow z \leq 0$). On the other side, if $x = 0$ then δ may be 1 or 0, that is, if nothing is delivered from the depot then the depot may be built or not.

If the condition must be imposed that the depot should *only* be built if some product are delivered from that depot, then equivalence is needed:

$$z > 0 \leftrightarrow \delta = 1$$

This can be modeled by the following two linear inequalities (supposing the lower bound for z is 0 and the minimal strictly positive quantity is ϵ):

$$z \leq M\delta , \quad z \geq \epsilon\delta$$

LPL generates the linear constraint:

$$C1: + z - 10 d \leq 0;$$

For equivalence the LPL code is ($\epsilon = 0.0001$):

$$\begin{aligned} C1a: + z - 10 d &\leq 0; \\ X14X: + z - 0.0001 d &\geq 0; \end{aligned}$$

Example 2: Suppose that z represents the quantity of a product manufactured at a marginal cost per unit of 5. If the product is manufactured at all there is an additional unique setup cost of 8. Hence, we need to distinguish two cases:

- (1) Either nothing is manufactured then $z = 0$, and cost = 0.
- (2) Or a quantity z is manufactured then $z > 0$, and cost = $5z + 8$

To model this situation, we introduce a Boolean (indicator) variable δ which is true if “something is manufactured” ($z > 0$), if nothing is manufactured the δ is false. Now we can link the indicator variable with the quantity z by the implication (meaning “if the quantity z is strictly larger than 0 then something is manufactured”):

$$z > 0 \rightarrow \delta \quad \text{or} \quad \bar{\delta} \rightarrow z \leq 0$$

This is modeled by the linear constraint (M being an upper bound for z): $z \leq M\delta$ and the cost function is formulated as: cost = $5z + 8\delta$

To verify the resulting model, we check for the values of δ :

- (1) if $\delta = 1$ (true) then we have: $z \leq U(z)$, cost = $5z + 8$
- (2) If $\delta = 0$ (false) then we have: $z \leq 0$, cost = 0

In LPL one may formulate the cost function directly – without needing to introduce explicitly a binary variable (although this is possible too) – as follows:

```
|   constraint C2: cost = 5*z + 8*(z>0);
```

It generates automatically the two linear constraints (where X13X is the additional binary variable):

```
|   C2: - 8 X13X - 5 z + cost = 0;
|   X14X: + z - 10 X13X <= 0;
```

Example 3: If z_A and z_B are the quantities in a blend of the two products A and B, then the logical formulation is: $z_A > 0 \rightarrow z_B \geq b$.

To model this situation, let's introduce an indicator variable δ with the meaning “A is included in the blend” that is: $z_A > 0 \rightarrow \delta$. And the requirement “if product A is included in the blend, then also at least b of product B must be included” can be modeled as: $\delta \rightarrow z_B \geq b$. The two constraints can be translated to mathematical inequalities as follows, where $U(z_A)$ is an upper bound for z_A and $L(z_B)$ is a lower bound for z_B :

$$\begin{aligned} z_A &\leq U(z_A) \cdot \delta \\ z_B &\geq L(z_B + b) \cdot \delta \end{aligned}$$

To verify the resulting model, we check for the values of δ :

- (1) If $\delta = 1$ (true) then $z_A \leq U(z_A)$, $z_B \geq b$ (supposing $L(z_B) = 0$)
(2) If $\delta = 0$ (false) then $z_A \leq 0$, $z_B \geq 0$

Example 4: A constraint A must hold if and only if the indicator variable is true. For example: $2v + 3w \leq 1 \leftrightarrow \delta$. This is equivalent to:

$$2v + 3w \leq 1 \rightarrow \delta , \quad \delta \rightarrow 2v + 3w \leq 1$$

Supposing $L(v) = L(w) = 0$, $U(v) = U(w) = 1$, $\epsilon = 0.001$ the resulting linear constraints are:

$$2v + 3w + 4\delta \leq 5 , \quad 2v + 3w + 1.001\delta \geq 1.001$$

Example 5: A general version of this exercise is explained in [30] Chap 3.5. We first present the solution given by Williams in [30]. Let's start with the constraint and repeat it:

$$\sum_j a_{1,j}x_j \leq b_1 \vee \sum_j a_{2,j}x_j \geq b_2 \leftrightarrow \sum_j a_{3,j}x_j \geq b_3 \vee \sum_j a_{4,j}x_j \leq b_4$$

Let the four top inequalities be P_i with $i \in \{1, \dots, 4\}$, as well as their lower and upper bound m_i, M_i with $i \in \{1, \dots, 4\}$. Then we have:

$$\begin{aligned} P_1 \vee P_2 &\leftrightarrow P_3 \vee P_4 && \text{which gives} \\ ((P_1 \vee P_2) \vee (\overline{P_3} \vee \overline{P_4})) \wedge ((\overline{P_1} \vee \overline{P_2}) \vee (P_3 \vee P_4)) &\text{ which gives} \\ (P_1 \vee P_2 \vee (\overline{P_3} \wedge \overline{P_4})) \wedge ((\overline{P_1} \wedge \overline{P_2}) \vee P_3 \vee P_4) \end{aligned}$$

Let's the indicator variables be δ_i with $i \in \{1, \dots, 4\}$, δ_5 , and δ_6 as follows:

$$\begin{aligned} \delta_i = 1 &\rightarrow P_i && \text{forall } i \in \{1, \dots, 4\} \\ \delta_5 = 1 &\rightarrow \overline{P_1} \\ \delta_5 = 1 &\rightarrow \overline{P_2} \\ \delta_6 = 1 &\rightarrow \overline{P_3} \\ \delta_6 = 1 &\rightarrow \overline{P_4} \end{aligned}$$

Hence, the top constraint reduces to:

$$(\delta_1 \vee \delta_2 \vee \delta_6) \wedge (\delta_3 \vee \delta_4 \vee \delta_5)$$

The 8 indicator variable definitions together with the Boolean constraint generated the following 10 linear inequalities:

$$\begin{aligned}
 \sum_j a_{1,j}x_j + (M_1 - b_1)\delta_1 &\leq M_1 \\
 \sum_j a_{2,j}x_j + (m_2 - b_2)\delta_2 &\leq m_2 \\
 \sum_j a_{3,j}x_j + (m_3 - b_3)\delta_3 &\leq m_3 \\
 \sum_j a_{4,j}x_j + (M_4 - b_4)\delta_4 &\leq M_4 \\
 \sum_j a_{1,j}x_j - (m_1 - b_1 + \epsilon)(1 - \delta_5) &\geq b_1 + \epsilon \\
 \sum_j a_{2,j}x_j - (M_2 - b_2 + \epsilon)(1 - \delta_5) &\geq b_2 - \epsilon \\
 \sum_j a_{3,j}x_j - (M_3 - b_3 + \epsilon)(1 - \delta_6) &\geq b_3 - \epsilon \\
 \sum_j a_{4,j}x_j - (m_4 - b_4 + \epsilon)(1 - \delta_6) &\geq b_4 + \epsilon \\
 \delta_1 + \delta_2 + \delta_6 &\geq 1 \\
 \delta_3 + \delta_4 + \delta_5 &\geq 1
 \end{aligned}$$

This is the translation given in [30]. In LPL, a somewhat different translation approach is chosen: *before* distributing and eliminating the equivalence (\leftrightarrow), the inequalities are defined by indicator variables:

$$\delta_i = 1 \rightarrow P_i \quad , \text{forall } i \in \{1, \dots, 4\}$$

Then the top constraint becomes:

$$\begin{aligned}
 \delta_1 \vee \delta_2 \leftrightarrow \delta_3 \vee \delta_4 &\quad \text{which gives} \\
 ((\delta_1 \vee \delta_2) \vee (\overline{\delta_3} \vee \overline{\delta_4})) \wedge ((\overline{\delta_1} \vee \overline{\delta_2}) \vee (\delta_3 \vee \delta_4)) &\quad \text{which gives} \\
 (\delta_1 \vee \delta_2 \vee (\overline{\delta_3} \wedge \overline{\delta_4})) \wedge ((\overline{\delta_1} \wedge \overline{\delta_2}) \vee \delta_3 \vee \delta_4) &\quad \text{which gives} \\
 (\delta_1 \vee \delta_2 \vee \overline{\delta_3}) \wedge (\delta_1 \vee \delta_2 \vee \overline{\delta_4}) \wedge (\overline{\delta_1} \vee \delta_3 \vee \delta_4) \wedge (\overline{\delta_2} \vee \delta_3 \vee \delta_4)
 \end{aligned}$$

The last line is an expression of four clauses, for which one can directly derive the four linear inequalities. Finally, this translation generates only 4 additional binaries and totally 8 linear constraints, in contrast with the translation given in [30] which generates 6 binaries and 10 linear constraints.

The code of LPL is as follows:

```

variable x[0..4]; y [0..5];
constraint C5: 3*x+4*y<=5 or 2*x+3*y>=4 <-> x+2*y>=3 or
           2*x+4*y<=5;

```

LPL generates the following 8 inequalities (together with the 4 binary variables X50X, X52X, X54X, X56X :

C5:	$+ X56X + X54X - X50X \geq 0;$
X51X:	$+ 4 y + 3 x + 27 X50X \leq 32;$
X53X:	$+ 3 y + 2 x - 4 X52X \geq 0;$
X55X:	$+ 2 y + x - 3 X54X \geq 0;$
X57X:	$+ 4 y + 2 x + 23 X56X \leq 28;$
X58X:	$+ X52X + X50X - X54X \geq 0;$
X59X:	$+ X56X + X54X - X52X \geq 0;$
X60X:	$+ X52X + X50X - X56X \geq 0;$

Questions

1. Check the output of LPL for example 5.

Answers

1. Taken the first definition: $\delta_1 = 1 \rightarrow P_1$, this gives

$$3x + 4y + (M_1 - b_1)\delta_1 \leq M_1$$

Since $b_1 = 5$ and $M_1 = 32$, this gives :

$$3x + 4y + 27\delta_1 \leq 32$$

corresponding to constraint X51X.

8.15. Boolean Expressions (logic1)

— Run LPL Code , HTML Document —

Problem: Study how the four following Boolean expressions A, B, C, D are translated into a mathematical 0-1-ILP formulation:

$$\begin{aligned} A &: x \wedge y \vee z \dot{\vee} w \\ B &: (x \wedge y \dot{\vee} z) \vee w \\ C &: \bigvee_{i \in I} (q_i \wedge p_i) \\ D &: \bigwedge_{i \in I} (q_i \vee p_i) \\ I &= \{1, \dots, n\}, n > 0 \end{aligned}$$

Modeling Steps:

Listing 8.8: The Complete Model implemented in LPL [72]

```
model Logic1 "Boolean Expressions";
parameter c:=4 "choose constraint";
set i:= [1..9];
binary variable x; y; z; w; u; p{i}; q{i};
constraint
  A if c=1: (x and y or z) xor w;
  B if c=2: (x and y xor z) or w;
  C if c=3: or{i} (p and q);
  D if c=4: and{i} (p or q);
solve 'nosolve';
Write('EQU'); //write the EQU-file
end
```

Expression A: $((x \wedge y) \vee z) \dot{\vee} w$

1. The first step is to apply a definition of the logical operator $\dot{\vee}$:

$$(x \dot{\vee} y) \equiv ((x \vee y) \wedge (\neg x \vee \neg y))$$

this gives:

$$((x \wedge y \vee z) \vee w) \wedge ((\neg(x \wedge y \vee z) \vee \neg w))$$

2. The first part gives $((x \vee z) \wedge (y \vee z) \vee w)$ which is:

$$(x \vee z \vee w) \wedge (y \vee z \vee w)$$

3. The second part gives $((\neg(x \vee y) \wedge \neg z) \vee \neg w)$ which is:

$$(\neg x \vee \neg y \vee \neg w) \wedge (\neg z \vee \neg w)$$

4. Putting the two parts together gives four clauses for A:

$$(x \vee z \vee w) \wedge (y \vee z \vee w) \wedge (\neg x \vee \neg y \vee \neg w) \wedge (\neg z \vee \neg w)$$

This is the CNF (conjunctive normal form) of the constraint A. Each single clause now can be formulated as a linear constraints as follows:

$$\begin{array}{ll} x + z + w & \geq 1 \\ y + z + w & \geq 1 \\ 1 - x + 1 - y + 1 - w & \geq 1 \\ 1 - z + 1 - w & \geq 1 \end{array}$$

Expression B: $((x \wedge y) \dot{\vee} z) \vee w$

1. Again, we apply the definition of $\dot{\vee}$. Hence the expression becomes:

$$(((x \wedge y) \vee z) \wedge ((\neg(x \wedge y) \vee \neg z)) \vee w$$

2. Redistributing \vee on the left and inverse moving \neg gives:

$$((x \vee z) \wedge (y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)) \vee w$$

3. Redistribute \vee an second time gives the following CNF:

$$(x \vee z \vee w) \wedge (y \vee z \vee w) \wedge (\neg x \vee \neg y \vee \neg z \vee w)$$

4. Again each single clause can be reformulated as a linear constraint:

$$\begin{array}{ll} x + z + w & \geq 1 \\ y + z + w & \geq 1 \\ 1 - x + 1 - y + 1 - z + w & \geq 1 \end{array}$$

Expression C: $\bigvee_{i \in I} (p_i \wedge q_i)$

The operator \bigvee is the indexed operator for “or”. The constraint is in fact (note that we have $I = \{1, \dots, n\}, n > 0$):

$$\underbrace{(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee \dots \vee (p_n \wedge q_n)}_{n \text{ times}}$$

1. Introduce a new binary variable r_i with $i \in I$ and add the constraints⁷

$$r_i \rightarrow (p_i \wedge q_i) , \text{ forall } i \in I$$

2. Substitute $p_i \wedge q_i$ by r_i in constraint C. This gives the system of the two constraint types:

$$\begin{aligned} & \bigvee_{i \in I} r_i \\ & r_i \rightarrow (p_i \wedge q_i) , \text{ forall } i \in I \end{aligned}$$

3. Replacing implication and distribute gives:

$$\begin{aligned} & \bigvee_{i \in I} r_i \\ & (\neg r_i \wedge p_i) \vee (\neg r_i \wedge q_i) , \text{ forall } i \in I \end{aligned}$$

4. This can be translated straight away into linear inequalities as:

$$\begin{aligned} \sum_{i \in I} r_i & \geq 1 \\ 1 - r_i + p_i & \geq 1 \quad \text{forall } i \in I \\ 1 - r_i + q_i & \geq 1 \quad \text{forall } i \in I \end{aligned}$$

An alternative of constraint C is not to introduce a new variable r_i , but to generate directly a CNF. However, the CNF would be exponential large in the size of set I. Hence, from a practical point of view, it is necessary to introduce a new binary variable r_i .

Expression D: $\bigwedge_{i \in I} (p_i \vee q_i)$

The operator \bigwedge is the indexed operator for “and”. The constraint is in fact (note that we have $I = \{1, \dots, n\}, n > 0$):

$$\underbrace{(p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge \dots \wedge (p_n \vee q_n)}_{n \text{ times}}$$

This constraint is already in the conjunctive normal form. Hence, the translation into linear inequalities is straightforward as:

$$p_i + q_i \geq 1 , \text{ forall } i \in I$$

Questions

⁷Note that in many cases an implication is all what is needed, however there are situations where an equivalence is imperatively required. LPL used implication whenever possible. If equivalence is needed the modeler must manually impose it.

1. Start LPL and look what it generates by looking at the EQU-file and the LPY-file.

Answers

1. the first constraint A generates the following linear constraints:

$$\begin{aligned} +x + z + w &\geq 1; \\ +w + z + y &\geq 1; \\ -x - y - w &\geq -2; \\ -w - z &\geq -1; \end{aligned}$$

The second constraint B produces the following linear constraints:

$$\begin{aligned} x &\geq 1-w-z \\ y &\geq 1-w-z \\ 1-x + 1-y + 1-z &\geq 1-w \end{aligned}$$

The third constraint C produces the following linear constraints:

$$\begin{aligned} C: \quad &+ X50[1] + X50[2] + X50[3] + X50[4] + X50[5] \\ &+ X50[6] + X50[7] + X50[8] + X50[9] \geq 1; \\ X51[1]: \quad &+ q[1] - X50[1] \geq 0; \\ X51[2]: \quad &+ q[2] - X50[2] \geq 0; \\ X51[3]: \quad &+ q[3] - X50[3] \geq 0; \\ X51[4]: \quad &+ q[4] - X50[4] \geq 0; \\ X51[5]: \quad &+ q[5] - X50[5] \geq 0; \\ X51[6]: \quad &+ q[6] - X50[6] \geq 0; \\ X51[7]: \quad &+ q[7] - X50[7] \geq 0; \\ X51[8]: \quad &+ q[8] - X50[8] \geq 0; \\ X51[9]: \quad &+ q[9] - X50[9] \geq 0; \\ X52[1]: \quad &+ p[1] - X50[1] \geq 0; \\ X52[2]: \quad &+ p[2] - X50[2] \geq 0; \\ X52[3]: \quad &+ p[3] - X50[3] \geq 0; \\ X52[4]: \quad &+ p[4] - X50[4] \geq 0; \\ X52[5]: \quad &+ p[5] - X50[5] \geq 0; \\ X52[6]: \quad &+ p[6] - X50[6] \geq 0; \\ X52[7]: \quad &+ p[7] - X50[7] \geq 0; \\ X52[8]: \quad &+ p[8] - X50[8] \geq 0; \\ X52[9]: \quad &+ p[9] - X50[9] \geq 0; \end{aligned}$$

where $X50\{i\}$ are the new binary variables introduced.

The fourth constraint D produces the following linear constraints:

$$\begin{aligned} D[1]: \quad &+ p[1] + q[1] \geq 1; \\ D[2]: \quad &+ p[2] + q[2] \geq 1; \\ D[3]: \quad &+ p[3] + q[3] \geq 1; \\ D[4]: \quad &+ p[4] + q[4] \geq 1; \\ D[5]: \quad &+ p[5] + q[5] \geq 1; \end{aligned}$$

```
D[6]: + p[6] + q[6] >= 1;  
D[7]: + p[7] + q[7] >= 1;  
D[8]: + p[8] + q[8] >= 1;  
D[9]: + p[9] + q[9] >= 1;
```

8.16. Math-Logic Expression III (logic2)

— Run LPL Code , HTML Document –

Problem: Model the following six Boolean constraints as linear inequalities (where $i \in I$):

$$\begin{aligned} C1 : d &\leftrightarrow (x \wedge y) \\ C2 : d &\leftrightarrow \bigwedge_{i \in I} z_i \\ D1 : d &\leftrightarrow (x \vee y) \\ D2 : d &\leftrightarrow \bigvee_{i \notin I} z_i \\ E1 : d &\leftrightarrow (x \veebar y) \\ F1 : d &\leftrightarrow (x \leftrightarrow y) \end{aligned}$$

Modeling Steps:

Listing 8.9: The Complete Model implemented in LPL [72]

```
model Logic2 "Math-Logic Expression III";
parameter c:=1 "choose constraint";
set i:=[1..9];
binary variable d; x; y; z{i};
constraint
  C1 if c=1: d <-> x and y;
  C2 if c=2: d <-> and{i} z;
  D1 if c=3: d <-> x or y;
  D2 if c=4: d <-> or{i} z;
  E1 if c=5: d <-> (x xor y);
  F1 if c=6: d <-> (x <-> y);
solve 'nosolve';
Write('EQU');
end
```

Constraint C1: Graphically, the constraint can be represented by a unit cube, where each of the 8 corners represent a true/false combination of the three variables (x, y, d) (see Figure 8.5). The expression C1 is true for the four black points and false otherwise. The black points can be separated from the others by two parallel planes defined by the points $\{(1, 1, 1), (0.5, 0, 1), (0, 0, 0)\}$ and $\{(0, 1, 0), (0.5, 1, 1), (0, 0, 1)\}$ (see Figure 8.5, left side)). The two planes can be formulated mathematically as: $x + y - 2d = 0$ and $x + y - 2d = 1$. Hence, the expression C1 could be expressed mathematically by the intersection of the two half-spaces :

$$0 \leq x + y - 2d \leq 1 \quad (\text{C1a})$$

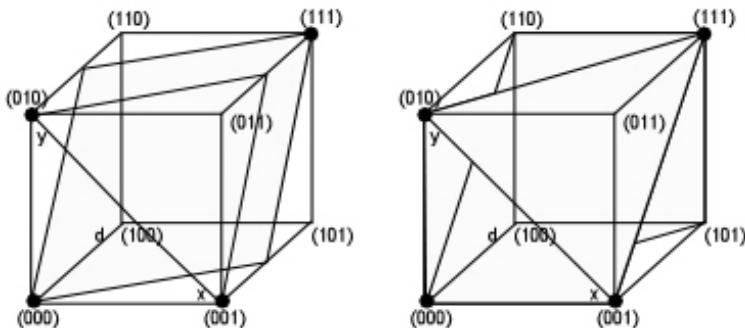


Figure 8.5: The Unit Cube with the 4 Feasible Points of C1

For a different formulation, consider the three planes defined by three points each as follows: $\{(010), (111), (001)\}$, $\{(111), (101), (000)\}$, and $\{(110), (000), (001)\}$ (see Figure 8.5, right side). These three planes are formulated mathematically as: $x + y - d = 1$, $x = d$, $y = d$. Hence, the constraint C1 could also be expressed mathematically by the intersection of the three half-spaces as follows:

$$\begin{aligned} x + y - d &\leq 1 \\ x &\geq d \quad (\text{C1b}) \\ y &\geq d \end{aligned} \tag{8.1}$$

Both mathematical formulations are correct. However, the second one with three linear inequalities is “better” because its LP-relaxation is tighter.⁸ LPL automatically generates the tighter formulation:

C1: $+ x - d \geq 0;$
X43X: $- y - x + d \geq -1;$
X44X: $+ y - d \geq 0;$

The transformation to a CNF (conjunctive normal form) is as follows:

$$\begin{aligned} d \leftrightarrow (x \wedge y) &\equiv (d \vee (\bar{x} \wedge \bar{y})) \wedge (\bar{d} \vee (x \wedge y)) \equiv \\ &(d \vee \bar{x} \vee \bar{y}) \wedge (\bar{d} \vee x) \wedge (\bar{d} \vee y) \end{aligned}$$

Constraint C2: In the same way as C1, this constraint C2 can be formulated by two parallel planes as follows (n is the cardinality of the set i):

⁸The LP-relaxation is obtained by replacing $x, y, d \in \{0, 1\}$ with $0 \leq x, y, d \leq 1$. “tighter” means that $\text{C1b} \subset \text{C1a}$ and this is “better” because most solver start with a LP-relaxation as initial solution. For a more profound theory see [42] or [51].

$$0 \leq \sum_{i=1}^n z_i - nd \leq n - 1$$

A tighter formulation is given by:

$$\begin{aligned} -\sum_{i=1}^n x_i + d &\geq 1 - n \\ x_i &\geq d \quad \text{forall } i \in \{1, \dots, n\} \end{aligned}$$

Constraint D1: Graphically, the constraint is true for the black points in the unit cube and false otherwise (see in Figure 8.6). These black points can be separated from the others by two parallel planes containing the points $\{(1, 1, 0), (0, 0, 0.5), (1, 0, 1)\}$ and $\{(1, 1, 1), (0.5, 0, 1), (0, 0, 0)\}$. The two planes are: $x + y - 2d = 0$ and $x + y - 2d = -1$. The intersection of the two half-spaces represents D1, it is:

$$-1 \leq x + y - 2d \leq 0$$

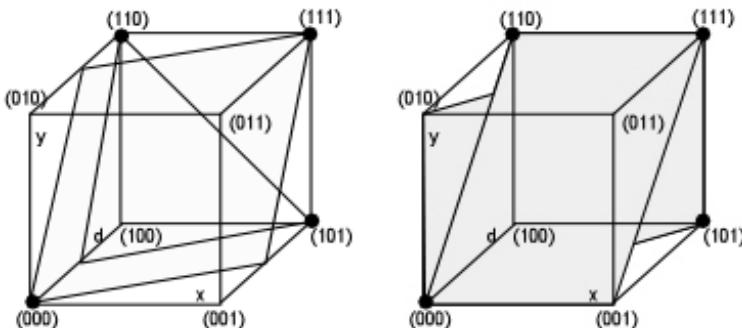


Figure 8.6: The Unit Cube with the 4 Feasible Points of D1

A tighter formulation is as follows:

$$\begin{aligned} x + y &\geq d \\ x &\leq d \\ y &\leq d \end{aligned}$$

The three corresponding planes are represented by the following coordinates in the (x, y, d) cube: $\{(110, 000, 101)\}, \{(111), (101), (000), (010)\}$, and $\{(110), (000), (001), (101)\}$ shown in Figure 8.6 (right side). LPL also generates the tighter form.

Constraint D2: Like D1 the constraint D2 corresponds to the two reformulations as follows:

$$-(n - 1) \leq \sum_{i=1}^n z_i \leq nd$$

And the tighter form:

$$\begin{aligned}\sum_{i=1}^n z_i &\geq d \\ -z_i + d &\geq 0 \quad \text{forall } i \in \{1, \dots, n\}\end{aligned}$$

Constraint E1: This constraint can be formulated in a tight form as:

$$\begin{array}{ll}x + y - d & \geq 0 \\x + y + d & \leq 2 \\x - y + d & \geq 0 \\-x + y + d & \geq 0\end{array}$$

Graphically the solution is given by the four corner points in Figure 8.7 (left side). Each half-space excludes a single corner of the unit cube.

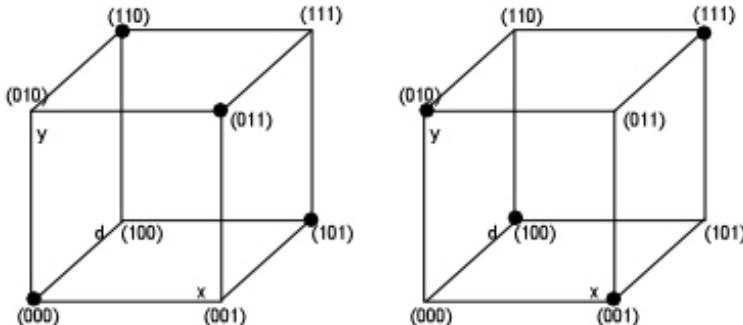


Figure 8.7: Four corner points

The four corresponding planes are defined by three points each as: $\{(000), (011), (110)\}$, $\{(110), (011), (101)\}$, $\{(000), (011), (101)\}$, and $\{(000), (101), (110)\}$.

Constraint F1: This constraint can be formulated in the tight form as:

$$\begin{array}{ll}-x + y - d & \geq -1 \\x - y - d & \leq -1 \\x + y + d & \geq 1 \\-x - y + d & \geq -1\end{array}$$

Graphically the solution is given by the four corner points in Figure 8.7 (right side).

The four planes can be constructed in the same way as in the previous example.

8.17. Indexed Boolean Expression (logic3)

— Run LPL Code , HTML Document –

Problem: Given 9 products $I = \{1, \dots, 9\}$, the following logical condition should be modeled: “If 3 or more out of the products $\{1, 2, 3, 4, 5\}$ are made, or less than 4 of products $\{3, 4, 5, 6, 8, 9\}$ are made then at least 2 of products $\{7, 8, 9\}$ must be made unless none of products $\{5, 6, 7\}$ are made.” This example has been published in [46].

Modeling Steps: To formulate this constraint, a Boolean variable P_i with $i \in I$ is introduced with the meaning “product i is made if $P_i = 1$ (that is, if P_i is true) otherwise $P_i = 0$ (or P_i is false)”. The logical statement can then be reformulated as follows:

“At least 3 of P_i with $i \in I_1 = \{1, 2, 3, 4, 5\}$ are true or at most 3 of P_i with $i \in I_2 = \{3, 4, 5, 6, 8, 9\}$ are true and not none of P_i with $i \in I_3 = \{5, 6, 7\}$ is true implies that at least 2 of P_i with $i \in I_4 = \{7, 8, 9\}$ are true”. Or formally:

$$((\text{atleast}(3)_{i \in I_1} P_i \vee \text{atmost}(3)_{i \in I_2} P_i) \wedge \neg(\text{nor}_{i \in I_3} P_i)) \rightarrow \text{atleast}(2)_{i \in I_4} P_i$$

Reducing implication and pushing not, gives:

$$\neg(\text{atleast}(3)_{i \in I_1} P_i \wedge \neg\text{atmost}(3)_{i \in I_2} P_i) \vee \text{nor}_{i \in I_3} P_i \vee \text{atleast}(2)_{i \in I_4} P_i$$

Applying some other rules for the indexed operators gives:

$$(\text{atmost}(2)_{i \in I_1} P_i \wedge \text{atleast}(4)_{i \in I_2} P_i) \vee \text{atleast}(0)_{i \in I_3} \neg P_i \vee \text{atleast}(2)_{i \in I_4} P_i$$

or:

$$\left(\sum_{i \in I_1} P_i \leq 2 \wedge \sum_{i \in I_2} P_i \geq 4 \right) \vee \bigwedge_{i \in I_3} \neg P_i \vee \sum_{i \in I_4} P_i \geq 2$$

Let us introduce three binary variables:

$$\begin{aligned}\delta_1 = 1 &\rightarrow \sum_{i \in I_1} P_i \leq 2 \\ \delta_1 = 1 &\rightarrow \sum_{i \in I_2} P_i \geq 4 \\ \delta_2 = 1 &\rightarrow \bigwedge_{i \in I_3} \neg P_i \\ \delta_3 = 1 &\rightarrow \sum_{i \in I_4} P_i \geq 2\end{aligned}$$

Hence, the whole constraint can be reduced to 5 linear inequalities as follows:

$$\begin{aligned}\sum_{i \in I_1} P_i + 3\delta_1 &\leq 5 \\ \sum_{i \in I_2} P_i - 4\delta_1 &\geq 0 \\ P_i + \delta_2 &\leq 1 \quad \text{forall } i \in I_3 \\ \sum_{i \in I_4} P_i - 2\delta_3 &\geq 0 \\ \delta_1 + \delta_2 + \delta_3 &\geq 1\end{aligned}$$

Listing 8.10: The Complete Model implemented in LPL [72]

```
model Logic3 "Indexed Boolean Expression";
set i := [1..9]           "A set of products";
i1{i} := [1 2 3 4 5]      "Sublist: products 1 to 5";
i2{i} := [3 4 5 6 8 9];
i3{i} := [5 6 7];
i4{i} := [7 8 9];
binary variable P{i} "Product i is produced?";
constraint Cond:
  (atleast(3){i1} P[i1] or atmost(3){i2} P[i2])
  and ~(nor{i3} P[i3]) -> atleast(2){i4} P[i4];
maximize obj: sum{i} P;
writeln(P);
end
```

Further Comments: LPL generates the following constraints, which corresponds exactly the solution above. It is also identical with the solution given in [46]:

```
Cond: X50X + X48X + X45X >= 1;
X46X: P[1] + P[2] + P[3] + P[4] + P[5] + 3 X45X <= 5;
X47X: P[3] + P[4] + P[5] + P[6] + P[8] + P[9] - 4 X45X >= 0;
X49X[5]: - X48X - P[5] >= -1;
```

```
X49X[6]: - X48X - P[6] >= -1;  
X49X[7]: - X48X - P[7] >= -1;  
X51X:   P[7] + P[8] + P[9] - 2 X50X >= 0;
```

Questions

1. For some reasons, the company wants only produce two products. Which product is certainly not in that list.

Answers

1. If only two product are produced then certainly product 5 and 6 cannot be part of them. Why? Because this makes the premise of the implication true and the consequence false, since then less than two of set I_4 are part and not none of set I_3 are in the part. One can check this by adding the two constraints (the model becomes infeasible – saying that there is no way to make this true):

```
| constraint D: sum{i} P <= 2;  
| constraint E: P[5] or P[6];
```

8.18. Disjunctive Constraint (logic4)

— Run LPL Code , HTML Document –

Problem: Model the disjunctive grey space defined in the Figure 8.8

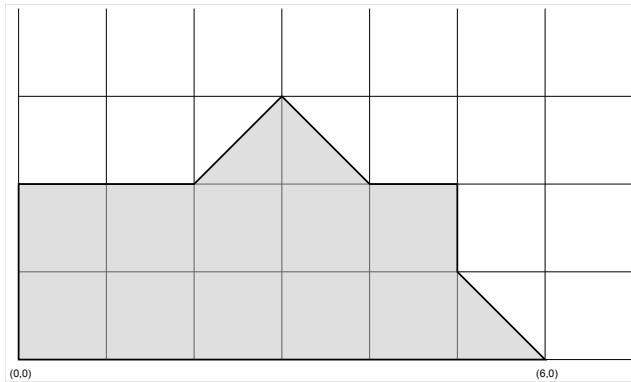


Figure 8.8: A Disjunctive Space

Modeling Steps: Informally, a set is concave, if some points on a straight segment with its endpoints in the set are outside the set. For a definition see Wikipedia. The grey set (space) in Figure 8.8 is concave (or disjunctive) and we know that this space cannot be modeled using a single LP – that is using linear inequalities, since the feasible space of an LP is always convex. A set of constraints can be interpreted as the *intersection* of all constraints in the set. In the Figure 8.8 the grey surface can be interpreted as the *union* of two convex figures: (1) a rectangle with the four corners $(0,0)$, $(0,2)$, $(5,2)$, and $(5,0)$. (2) a triangle with the corners $(0,0)$, $(3,3)$, and $(6,0)$.

1. We introduce two variables $x \geq 0$ (horizontal extension) and $y \geq 0$ (vertical extension).
2. The rectangle space R can be modeled as a convex space (intersection of two linear constraints):

$$R : x \leq 6 \wedge y \leq 2$$

3. The triangle space T can be modeled as a convex space:

$$T : y \leq x \wedge y \leq 6 - x$$

4. The union of R and T is therefore:

$$(x \leq 6 \wedge y \leq 2) \vee (y \leq x \wedge y \leq 6 - x)$$

Listing 8.11: The Complete Model implemented in LPL [72]

```

model Logic4 "Disjunctive Constraint";
  variable x[0..10]; y[0..20];
  constraint Grey: (x<=5 and y<=2) or (y<=x and y<=6-x);
  maximize obj: x+y;
  Write('Optimum is: (%d,%d)\n' ,x,y);
end

```

Solution: Depending on the objective function, various optimal points are found and they are easy to verify :

1. With the function $\max x + y$, the optimum is $(5, 2)$.
2. With the function $\max y$, the optimum is $(3, 3)$.
3. With the function $\max x$, the optimum is $(6, 0)$.
4. With the function $\max y - x$, the optimum is $(0, 2)$.

LPL transforms the disjunctive constraint into the following 5 linear constraints, adding two binary variables x_{40} and x_{43} :

max:	$+ y + x;$
Grey:	$+ X43X + X40X \geq 1;$
X41X:	$+ x + 5 X40X \leq 10;$
X42X:	$+ y + 18 X40X \leq 20;$
X44X:	$- x + y + 20 X43X \leq 20;$
X45X:	$+ x + y + 24 X43X \leq 30;$

8.19. Two Liquid Containers Problem (logics5)

— Run LPL Code , HTML Document –

Problem: A company produces two liquids A and B at unknown quantities x and y . The liquids are stored in two containers of capacity a and b . The two liquids can be stored in both containers, but cannot be mixed in the same container. Normally, liquid A is stored in the first and liquid B in the second container. The company could therefore produce the maximum quantity a of liquid A and a maximum quantity b of liquid B. But in some situations, it may be more advantageous to produce more liquid of the same type and none of the other. In this case, the company may produce one liquid at a maximum quantity of $a + b$ and to store it in both containers. Hence, either the company must produce both liquids at quantities less than a and b respectively, or it must produce only one liquid with quantity less than $a + b$ (see Figure 8.9). Formulate the capacity constraint (This problem is from [47]).

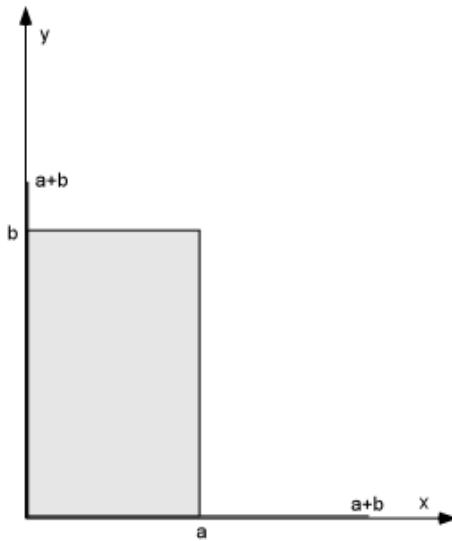


Figure 8.9: The Feasible Space of the Liquid Container Problem

Modeling Steps: This is a typical very simple problem of a *disjunctive set*.

1. We introduce two variables for the quantities of the two liquids: x and y .
2. Three cases can be distinguished: Either we produce only the first liquid A and store it in both containers, or we produce only the second liquid B and store it also in both containers or we produce both then we have to store the first in one and the second in the second container separately. The feasible space is shown in Figure 8.9.

- First case: $y = 0$ and $x \leq a + b$ segment $(0, 0)$ to $(a + b, 0)$
 Second case: $x = 0$ and $y \leq a + b$ segment $(0, 0)$ to $(0, a + b)$
 Third case: $x \leq a$ and $y \leq b$ rectangle $(0, 0)$ to (a, b)

3. At least one of the three cases must be true, hence:

$$(y = 0 \wedge x \leq a + b) \vee (x = 0 \wedge y \leq a + b) \vee (x \leq a \wedge y \leq b)$$

4. It is easy to verify that this expression corresponds also to (we suppose that $x, y \geq 0$):

$$(x > a \rightarrow y \leq 0) \wedge (y > b \rightarrow x \leq 0)$$

Listing 8.12: The Complete Model implemented in LPL [72]

```
model Logic5 "Two Liquid Containers Problem";
parameter a:=10; b:=12;
variable x [0..a+b]; y [0..a+b];
constraint capa: (x>a -> y<=0) and (y>b -> x<=0);
--constraint capa: y=0 and x<=a+b or x=0 and y<=a+b or
-- x<=a and y<=b;
maximize obj:x+y;
Writep(x,y);
end
```

Solution: The logical capacity constraint can be translated into the following linear constraints by adding two binary variables p and q :

$$\begin{aligned} x - a &\leq b \cdot p \\ y - b &\leq a \cdot q \\ x &\leq (a + b) \cdot (1 - q) \\ y &\leq (a + b) \cdot (1 - p) \\ p, q &\in \{0, 1\}, 0 \leq x, y \leq a + b \end{aligned}$$

Further Comments: To translate the logical constraint into the linear inequalities, do the following:

1. Start with the expression:

$$(x > a \rightarrow y \leq 0) \wedge (y > b \rightarrow x \leq 0)$$

2. Replacing the implication gives:

$$(x \leq a \vee y \leq 0) \wedge (y \leq b \vee x \leq 0)$$

3. Add four binary variables and substitute the four subexpressions⁹:

$$\begin{array}{ll} p_1 = 1 & \rightarrow \quad x \leq a \\ p_2 = 1 & \rightarrow \quad y \leq 0 \\ q_1 = 1 & \rightarrow \quad y \leq b \\ q_2 = 1 & \rightarrow \quad x \leq 0 \\ p_1, p_2, q_1, q_2 \in \{0, 1\} & \end{array}$$

4. The expression becomes :

$$(p_1 \vee p_2) \wedge (q_1 \vee q_2)$$

5. which is :

$$p_1 + p_2 \geq 1 \quad , \quad q_1 + q_2 \geq 1$$

6. Translating the four definitions into linear constraints gives (where $U(\alpha)$ is an upper bound for α):

$$\begin{array}{lcl} x - a & \leq & U(x - a) \cdot (1 - p_1) \\ y & \leq & U(y) \cdot (1 - p_2) \\ y - b & \leq & U(y - b) \cdot (1 - q_1) \\ x & \leq & U(x) \cdot (1 - q_2) \end{array}$$

7. Since $U(x) = U(y) = a + b$ we have the following model:

$$\begin{array}{lcl} p_1 + p_2 & \geq & 1 \\ q_1 + q_2 & \geq & 1 \\ x - a & \leq & b \cdot (1 - p_1) \\ y & \leq & (a + b) \cdot (1 - p_2) \\ y - b & \leq & a \cdot (1 - q_1) \\ x & \leq & (a + b) \cdot (1 - q_2) \end{array}$$

8. A further reduction can be obtained by observing that p_2 can be replaced by $1 - p_1$, and q_2 by $1 - q_1$ and the solution follows.

LPL translates the code

```
parameter a:=10; b:=12;
variable x [0..a+b]; y [0..a+b];
constraint capa: (x>a -> y<=0) and (y>b -> x<=0);
```

into the following linear constraints:

⁹In this case we substitute the expressions by an implication: we force the subexpression to be true if the binary variable is true, if the binary variable is false than nothing is forced. If we want also force the expression to be false if the binary is false then we must replace the implication by equivalence.

```

capa: + x - 12*X39X <= 10;
X42X: + y - 10*X41X <= 12;
X43X: + y + 22*X39X <= 22;
X44X: + x + 22*X41X <= 22;
-- bounds --
x <= 22;
y <= 22;
X39X <= 1 , binary;
X41X <= 1 , binary;

```

This is exactly what we have obtained above. Note, however, that the linear inequalities are logically not equivalent with the logical constraint, but they are *implied*.

Questions

1. Verify that the two logical expressions above are equivalent.
2. The logical constraint is translated into a mixed integer formulation as given above. Show that this linear formulation is correct.
3. What happens if the first liquid A generates 10 times the profit of the second and we want to maximize profit?

Answers

1. In the expression

$$(x > a \rightarrow y \leq 0) \wedge (y > b \rightarrow x \leq 0)$$

replacing $a \rightarrow b$ by $\bar{a} \vee b$ gives:

$$(x \leq a \vee y \leq 0) \wedge (y \leq b \vee x \leq 0)$$

Redistributing \vee gives:

$$(x \leq a \wedge y \leq b) \vee (x \leq a \wedge y \leq 0) \vee (y \leq b \wedge x \leq 0) \vee (y \leq 0 \wedge x \leq 0)$$

Now the last term is contained in all the previous ones, so it can be removed and the formula follows.

2. Suppose $p = 0$ and $q = 0$, then $x \leq 10, y \leq 12$, that is, the two containers are used separately to store the two liquids.

Suppose $p = 1$ and $q = 0$, then $x \leq 22, y \leq 0$, that is, both containers are used to store the first liquid and the second liquid is not produced.

Suppose $p = 0$ and $q = 1$, then $x \leq 0, y \leq 22$, that is, both containers are used to store the second liquid and the first liquid is not produced.

Suppose $p = 1$ and $q = 1$ then $x \leq 0, y \leq 0$. This case will not be realised since at least one (x or y) is maximized. Hence, the model correctly reflects the three cases discussed above.

3. We change the maximizing function to `maximize obj:10*x+y;`. Solving the model then generates the solution $x = 22$ and $y = 0$.

8.20. Logical Conditions in an LP (logic6)

— Run LPL Code , HTML Document –

Problem: In a food blending problem, where several ingredients are mixed to produce a food, the modeler wants – in addition to the common mathematical balancing and capacity restrictions – to impose the following logical constraints:

1. The food may never be made up of more than three oils (ingredients).
2. If an oil is used at least 20 tons and at most 200 tons must be used.
3. If either Veg1 or Veg2 is used (two vegetal oils) then Oil3 (a non-vegetal oil) must also be used in the blend.

How can these conditions be formulated mathematically? This model part belongs to a model presented in [31]

Modeling Steps:

1. For each ingredient $i \in I$ there is a variable $x_i \geq 0$ which is the quantity of the ingredient.
2. For each ingredient i a binary variable d_i is introduced with the meaning “there is (or is not) a certain quantity of the ingredient i in a blend (that means, zero quantity (for $d_i = 0$) or at least 20 and at most 200 (for $d_i = 1$)).
3. The binary variable d_i is linked to x_i in the following way: $d_i \rightarrow (20 \leq x_i \leq 200)$ with $i \in I$, that is, if d_i is true then x_i must be between 20 and 200.

Listing 8.13: The Complete Model implemented in LPL [72]

```

model Logic6 "Logical Conditions in an LP";
set i := /Veg1 Veg2 Oil11 Oil12 Oil13/  "Ingredients in a
blend";
parameter
  Lo:=20 "Lower bound in use";
  Up:=200 "Upper bound in use";
variable x{i} [0..300] "Quantity of ingredient used in
the blend";
binary variable d{i}; //: Lo <= x <= Up;
constraint
  Cond1: sum{i} d <= 3;
  Cond2{i}: d -> (Lo <= x <= Up);
  Cond3: d['Veg1'] or d['Veg2'] -> d['Oil3'];
solve; //minimize obj: sum{i} x "Minimize the output";
//Writep(obj, x, d);

```

end

Note that the lower and upper bounds of x_i is not the same as the minimal and maximal quantity *used in the blend*. Therefore, the second condition is translated into:

$$\begin{aligned} x_i &\leq 20\delta_i && \text{forall } i \in I \\ x_i + 100\delta_i &\leq 300 && \text{forall } f_i \in I \end{aligned}$$

Further Comments: In LPL the second condition can also be integrated in the declaration of the binary variable as follows;

binary variable d{i}: Lo <= x <= Up

In this case, the assignment is interpreted as an implication. In both cases, LPL generates the following linear model:

```

Cond1: + d[Veg1] + d[Veg2] + d[Oil1] + d[Oil2] + d[Oil3] <= 3;
Cond2[Veg1]: - x[Veg1] + 20 d[Veg1] <= 0;
Cond2[Veg2]: - x[Veg2] + 20 d[Veg2] <= 0;
Cond2[Oil1]: - x[Oil1] + 20 d[Oil1] <= 0;
Cond2[Oil2]: - x[Oil2] + 20 d[Oil2] <= 0;
Cond2[Oil3]: - x[Oil3] + 20 d[Oil3] <= 0;
Cond3: - d[Veg1] + d[Oil3] >= 0;
X45X[Veg1]: + x[Veg1] + 100 d[Veg1] <= 300;
X45X[Veg2]: + x[Veg2] + 100 d[Veg2] <= 300;
X45X[Oil1]: + x[Oil1] + 100 d[Oil1] <= 300;
X45X[Oil2]: + x[Oil2] + 100 d[Oil2] <= 300;
X45X[Oil3]: + x[Oil3] + 100 d[Oil3] <= 300;
X46X: - d[Veg2] + d[Oil3] >= 0;
  
```

8.21. Math-Logic Expression II ([logic7](#))

— Run LPL Code , HTML Document —

Problem: Formulate the following constraint: “If at most 2 kg of the ingredient A or at most 3 kg of the ingredient B is in a mixture then at most 8 kg of the ingredient E or at least 7 kg of the ingredient F is in the mixture and vice versa.” This example is from [46].

/MOModeling Suppose the quantity of kilograms of each ingredient is a , b , e , and f , then this statement can be formulated as follows:

$$a \leq 2 \vee b \leq 3 \leftrightarrow e \leq 8 \vee f \geq 7$$

One can apply exactly the same procedure as in model [logic0](#), Example 5.

Listing 8.14: The Complete Model implemented in LPL [[72](#)]

```
model Logic7 "Math-Logic Expression II";
  variable a [1..20]; b [0..100]; e[2..20]; f[3..20];
  constraint S: a<=2 or b<=3 <-> e<=8 or f>=7;
  minimize obj: a+b+e+f;
  Writep(obj);
end
```

Further Comments: McKinnon ([46]) give a solution with 6 additional indicator variables and 10 linear constraints, while LPL generates a model with 4 additional binaries and 8 linear constraints as follows:

```
S: + X48X + X46X - X42X >= 0;
X43X: + a + 18 X42X <= 20;
X45X: + b + 97 X44X <= 100;
X47X: + e + 12 X46X <= 20;
X49X: + f - 4 X48X >= 3;
X50X: + X44X + X42X - X46X >= 0;
X51X: + X48X + X46X - X44X >= 0;
X52X: + X44X + X42X - X48X >= 0;
```

Questions

1. Change $e \leq 8$ to $e \geq 8$ and compare the two objective values, What do you notice?

Answers

1. The solution to the first model is 6 while to the second it is 10. Minimizing the sum of the four variables will set them to their lower bound,

that is: $a = 1$, $b = 0$, $e = 2$, and $f = 3$, since the left side and the right side of the equivalence are both true, the constraint is fulfilled.

In the second case, the right side is true only if $f \geq 7$, hence the solution is $a + b + c + d = 10$. If, on the other hand, the left would be false then $a \geq 3$ and $b \geq 4$ must hold and f and g would be at there lower bound, hence $a + b + c + d \geq 12$ which is larger than 10.

8.22. Transformations of logical statements ([logic8](#))

— Run LPL Code , [HTML Document](#) —

Problem: A list of rules

Listing 8.15: The Complete Model implemented in LPL [[72](#)]

```

model logic8 "Transformations of logical statements";
set i := [1..5];
      j := [1..10];
parameter a;
binary variable
  x{i}; xx{i}; xxx{i,j}; xxxx{j}; v; w; y; z; zz;
real variable
  p [0..10]; r[0..10]; rr [10..30]; --u; o; q{i}; t{i};
  tt{j,i};
alldiff variable
  d{i} [1..5];
constraint
  -- T0 rules
  Rule01 : y nor z;
  Rule02 : y nand z;
  Rule03 : and{i} x;
  Rule04 : or{i} x;
  Rule05 : xor{i} x;
  Rule06 : nor{i} x;
  Rule07 : nand{i} x;
  Rule08 : y <= z <= p;
  Rule08a : y <= z <= p <= v;
  -- T1 rules
  Rule10 : y->z;
  Rule11 : z<-y;
  Rule12 : forall{i} x;
  Rule13 : exist{i} x;
  Rule14 : atleast(-2){i} x;
  Rule15 : atmost(-2){i} x;
  Rule16 : exactly(0){i} x;
  -- T2 rules:
  Rule20 : ~(~y);
  Rule21 : ~(y and z);
  Rule22 : ~(y or z);
  Rule23 : ~(y<->z);
  Rule24 : ~(y xor z);
  Rule25 : ~(y<z);
  Rule26 : ~(atleast(3){i} x);
  Rule26a : ~(atleast(0){i} x); // 0 means #
  Rule27 : ~(atmost(2){i} x);
  Rule28 : ~(exactly(3){i} x);
  Rule29 : 0 <> y;
```

```
Rule29a : 0=y;
Rule30  : y = z;
Rule31  : y <-> z;
Rule32  : y <> z;
Rule33  : y xor z;
Rule34  : exactly(2){i} x;
Rule35  : atmost(2){i} x;
Rule33a : atmost(0){i} x;
Rule36  : atleast(2){i} x;
-- T3 rules
Rule40  : z or (v and y);
Rule40a : (v and y) or z;
Rule41  : y or atleast(0){i} x;
Rule41a : atleast(0){i} x or y;
solve;
end
```

8.23. Importing Energy (**import**)

— Run LPL Code , HTML Document —

Problem: Coal, gas and nuclear fuel can be imported in order to satisfy the energy demands of a country. Three grades of gas and coal (low, medium, high) and one grade of nuclear fuel may be imported. The costs are known. Furthermore, there are upper and lower limits in the imported quantities from a country. What quantities should be imported if the import cost have to be minimized? (The model is from [49]). In addition, three additional conditions must be fulfilled:

1. A supply condition: Each country can supply either up to three (non-nuclear) low or medium grade fuels or nuclear fuel and one high grade fuel.
2. Environmental restriction: Environmental regulations require that nuclear fuel can be used only if medium and low grades of gas and coal are excluded.
3. Energy mixing condition: If gas is imported then either the amount of gas energy imported must lie between 40-50 percent and the amount of coal energy must be between 20-30 percent of the total energy imported or the quantity of gas energy must lie between 50-60 percent and coal is not imported.

Modeling Steps: Three index-set define are defined: $i \in I = \{\text{low}, \text{medium}, \text{high}\}$ the grade fuel, $j \in J = \{\text{coal}, \text{gas}\}$ the non-nuclear energy sources, and $k \in K = \{\text{GR}, \text{FR}, \text{IC}\}$ the countries from where to import. Nuclear energy is treated apart because it come only in one grade.

1. The data are defined and explained in the model code (see parameter section).
2. We first introduce two continuous variables: the quantity of non-nuclear fuel ($x_{i,j,k}$) and the quantity of nuclear fuel (y_k).
3. The sum of all imports must be the desired energy amount e . Hence we have

$$\sum_{i,j,k} x_{i,j,k} + \sum_k y_k = e$$

4. The import costs are to be minimized. Hence:

$$\min \sum_{i,j,k} c_{i,j,k} x_{i,j,k} + \sum_k n c_k y_k)$$

5. The three logical requirements are modeled as follows. Boolean predicates (binaries) are introduced to link the mathematical with the logical part of the model. Let $P_{i,j,k}$ be a predicate with the definition: "If at least $l_{i,j,k}$ or at most $u_{i,j,k}$ of non-nuclear energy is imported then $P_{i,j,k}$ ". For the nuclear energy a similar definition is introduced: "If at least nl_k or at most nu_k of nuclear energy is imported then N_k ". We have:

$$\begin{aligned} P_{i,j,k} &\leftarrow (l_{i,j,k} \leq x_{i,j,k} \leq u_{i,j,k}) && \text{forall } i, j, k \\ N_k &\leftarrow (nl_k \leq y_k \leq nu_k) && \text{forall } k \end{aligned}$$

6. In condition 3, two other predicates are needed: "If the total amount of non-nuclear imported fuel, lays between 40-50% for gas and 20-30% for coal of the total imported non-nuclear fuel then Q_j ", and "if imported gas is between 50-60% of the total imported non-nuclear fuel then R ". Hence:

$$\begin{aligned} Q_j &\leftarrow (eL_j \leq \sum_{i,k} x_{i,j,k} \leq eU_j) && \text{forall } j \\ R &\leftarrow (eA \leq \sum_{i,k} x_{i,'gas',k} \leq eB) \end{aligned}$$

7. Using these predicates, the three conditions are formulated as follows:

- "For each country k , either at most 3 of $P_{i,j,k}$ (with $i \neq \text{'high'}$) are true or N_k is true and exactly one $P_{\text{'high'},j,k}$ is true"
- "At least one N_k is true, only if none of $P_{i,j,k}$ is true (with $i \neq \text{'high'}$)" and
- "If any of $P_{i,'gas',k}$ is true, then either Q_j is true or R is true and none of $P_{i,'coal',k}$ is true."

Some remarks on the formulations

- The first condition has the form: "for each k , either A or B ". Normally, "either ... or" is translated as exclusive or, but sometimes we just mean or (non-exclusive). It depends on the context. Here we want to say that A and B cannot both be true at the same time. Therefore we have: $A \dot{\vee} B$.
- The second condition has the form: " A only if B " which is simply $A \rightarrow B$.
- The third condition has the form: "If A then (either B or (C and D))", which is $A \rightarrow (B \dot{\vee} (C \wedge D))$.

Listing 8.16: The Complete Model implemented in LPL [72]

```
model ImportEnergy "Importing Energy";
set
  i   := [low med high] "Quality grades of energy";
  j   := [gas, coal]     "Non-nuclear energy sources";
```

```

k := [GB FR IC]      "Countries exporting energy";
parameter
c{i,j,k} "Non-nuclear energy cost";
l{i,j,k} "Low bound on the quan of non-nucl energy";
u{i,j,k} "Upp bound on the quan of non-nucl energy";
nc{k} "Nuclear energy cost";
nl{k} "Low bound on the quantity of nucl energy";
nu{k} "Upp bound on the quantity of nucl energy";
e "Desired energy amount to import";
L{j} "Min perc of energy if coal+gas imported";
U{j} "Max perc of energy if coal+gas imported";
A "Min perc of gas if gas only is imported";
B "Max perc of gas if gas only is imported";
variable
x{i,j,k} [0..u] "Quant. of non-nucl energy imported";
y{k} [0..nu] "Quantity of nucl energy imported";
binary variable
P{i,j,k} <- l<=x;
N{k} <- nl<=y;
Q{j} <- e*L <= sum{i,k} x <= e*U;
R <- e*A <= sum{i,k} x[i,'gas',k] <= e*B;
constraint
ImportReq : sum{i,j,k} x + sum{k} y = e;
C1{k}: atmost(3){i,j|i>'high'} P xor (N and
    xor{j} P['high',j,k]);
C2: or{k} N -> nor{i,j,k|i>'high'} P;
C3: or{i,k}P[i,'gas',k] -> and{j}Q xor R and
    nor{i,k}P[i,'coal',k];
minimize Cost: sum{i,j,k} c*x + sum{k} nc*y;
Writep(Cost,x,y,P,N,Q,R);
model data;
e := 12000;
c{i,j,k} := Rnd(1,4); nc{k} := Rnd(1,3);
l{i,j,k} := Rnd(10,40); u{i,j,k} := Rnd(1000,2000);
nl{k} := Rnd(20,30); nu{k} := Rnd(1000,5000);
L{j} := [0.20 0.40]; U{j} := [0.30 0.50];
A := 0.50; B := 0.60;
end
end

```

8.24. Assembling a Radio (radio)

— Run LPL Code , HTML Document –

Problem: To assemble a radio any of three types (T1,T2,T3) of tubes can be used. The box may be either of wood W or of plastic material P. When using P, then dimensionality requirements impose the choice of T2 and a special power supply F (since there is no space for a transformer S). T1 needs F. When T2 or T3 is chosen then we need S and not F. The price of each component is given. A radio made of wood or of plastic has different selling prices. Should we build a radio of wood or of plastic when we want to maximize profit? (The problem is from [53]).

Modeling Steps: This model shows how mathematical and logical constraints can be used side by side.

1. There are 7 possible components that are used to build a radio. For each component a binary variable is introduced, that says whether to use it or not to build our radio.
2. The constraints easy to derive. The objective function is the selling prices minus the costs of all components. Since the radio is either of wood or plastic only one of the variable W or P is different from zero.

Listing 8.17: The Complete Model implemented in LPL [72]

```

model Radio "Assembling a Radio";
binary variable
  T1 "Tube of type I";
  T2 "Tube of type II";
  T3 "Tube of type III";
  W "a wooden box";
  P "a plastic box";
  F "a transformator";
  S "a special power supply";
constraint
  R1: T1+T2+T3=1      "Use one tube";
  R2: W xor P        "Wood or plastic?";
  R3: P -> T2 and S  "If Plastic then use T2 and S";
  R4: T1 -> F        "T1 needs a transformator";
  R5: T2 or T3 -> S "When choosing T2 or T3, we need S
    ";
  R6: F xor S        "Either F or S must be used";
expression cost:=55*T1+58*T2+56*T3+25*F+23*S+9*W+6*P
  +10;
maximize Profit: 110*W + 105*P - cost;
  Write('The profit is: %d, the radio is made of \
    %s.\n',Profit,if(W,'wood','plastic'));
end

```

Solution: The profit for one radio made of wood is 12. It uses tube III and a special power supply.

Questions

1. What is the profit if the radio is made of plastic?

Answers

1. The profit is 8. It uses tube II and also a special power supply. This can be found by adding the constraint:

```
|   constraint R7: ~W;    //do not use a wooden box !
```

8.25. Pigeonhole Problem (**pihole**)

— Run LPL Code , HTML Document –

Problem: The pigeon hole problem is to place $n + 1$ pigeons in n holes so that no hole contains more than one pigeon. The problem is clearly infeasible. The problem is interesting for its concise formulation as a SAT (satisfiability) problem and because it is very difficult to solve using the method of resolution.

A SAT formulation is (with $x_{i,j} = \{\text{true}, \text{false}\}$):

$$\begin{aligned} \bigvee_j^n x_{i,j}, \quad & \text{forall } i = 1, \dots, n + 1 \\ \neg x_{i,j} \vee \neg x_{k,j}, \quad & \text{forall } i, k = 1, \dots, n + 1, i \neq k, j = 1, \dots, n \end{aligned}$$

Formulated as a IP linear problem, it can be *very easy* or *very difficult* to prove infeasibility, depending on how it is formulated and depending on the solvers methods.

The problem was described in (see [11], p.110, and [6], p.13).

Modeling Steps:

1. We introduce a binary variable for each (i, j) -combination, that is, $x_{i,j} = 1$ if pigeon i is in hole j , otherwise $x_{i,j} = 0$.
2. The first constraint states that each pigeon is placed in exactly one hole.

$$\sum_{j=1}^n x_{i,j} = 1, \quad \text{forall } i = 1, \dots, n + 1$$

3. The second constraint states that for each pair of pigeons $(i, k), i \neq k$, both do not occupy the same hole j (constraint S0):

$$x_{i,j} + x_{k,j} \leq 1, \quad \text{forall } i, k = 1, \dots, n + 1, i \neq k, j = 1, \dots, n$$

4. There is another way to formulate this last constraint: For each hole j there can be at most one pigeon (constraint S1):

$$\sum_i^{n+1} x_{i,j} \leq 1, \quad \text{forall } j = 1, \dots, n$$

Listing 8.18: The Complete Model implemented in LPL [72]

```
model Pihole "Pigeonhole Problem";
```

```
--SetSolver(glpkSol);
parameter n:=10 "number of holes";
set i,k:=1..n+1 "n+1 Pigeons";
set j :=1..n "n Holes";
binary variable x{i,j};
constraint
  R{i}: sum{j} x = 1 "Each pigeon i must be in exactly
    one hole";
--S0{i,k,j|i<>k}: x[i,j] + x[k,j] <= 1;
  S1{j}: sum{i} x <= 1;
solve;
end
```

Note that the first formulation (with constraint S0) has $n(n + 1)^2$ constraint, while the second formulation (with constraint S1 instead) is much smaller and contains only $2n + 1$ constraint. Furthermore, the LP relaxation of the first formulation is feasible with $x_{i,j} = 1/n$, while the LP relaxation of the second formulation is infeasible. Thus, IP solvers – based on LP relaxations – will immediately discover the infeasibility of the pigeon hole problem in the second formulation, while in the first formulation the infeasibility could be very difficult to be proven. Therefore, the free solver *glpk* found the feasibility of the second formulation immediately, while it was not possible to prove infeasibility for the first formulation in reasonable time.

Further Comments: Alternatively, the constraints can also be formulated in a purly logical way in LPL as follows:

```
constraint
  R{i}: exactly(1){j} x  "Each pigeon i must be in
    exactly one hole";
--S0{i,k,j|i<>k}: x[i,j] nand x[k,j];
  S1{j}: atmost(1){i} x;
```

Questions

1. Try to solve both model versions – once with S0 and without S1 and once with S1 and without S0, with the two solvers *glpk* and *lp_solve*.
2. Try a problem with $n = 100$ and solve it with a commercial solver – like Gurobi. Try both formulations.

Answers

1. Prove infeasibility with S1 is easy and the solver takes no time. With S1 the solvers take a long time to prove infeasibility.
2. Depending on commercial solvers, it is easy or difficult to solve with S0. For Gurobi it was easy to solve it.

MY VISION

“When I am working on a problem I never think about beauty. I think only how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong.”

— R. Buckminster Fuller

“By relieving the brain of unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race. Before the introduction of the Arabic notation, multiplication was difficult, and division even of integers called into play the highest mathematical faculties. Probably nothing in the modern world would have more astonished a Greek mathematician than to learn that ... a huge proportion of the population of Western Europe could perform the operation of division for the largest numbers. This faculty would have seemed to him a sheer impossibility ... Our modern power of easy reckoning with decimal fractions is the almost miraculous result of the gradual discovery of the perfect notation.”

— A. Whitehead

“The right tool for the right job!”

— Common sense

This paper is an update of my paper “Modeling Languages in Optimisation: A new Paradigm of Programming” [77] published 25 years ago. It exposed my ideas of why we need a new programming language paradigm. In the meantime a lot has happened: The paradigm of *constraint programming* has been established, new “packages” in mathematical modeling in modern programming languages, as Python, Julia, C++, a.o., have popped up recently, several commercial modeling systems are on the market, such as AIMMS, MOSEL, HEXALY, and several algebraic modeling languages, as AMPL, GAMS, LINGO, etc. have been extended.

I have contributed with my own modeling language, namely LPL. At the beginning of my career as a researcher, I implemented LPL (Linear Programming Language) as a tool to formulate several larger LPs (Linear Programs), which we used at the Department of Informatics at the University of Fribourg for various real-life projects. Soon I discovered that this kind of language notation could be used for many different other applications. I added and removed many features to the language, always on the quest to find out what is the “best” (simplest, shortest, most readable, efficient) way to formulate and to model a concrete problem as a mathematical model. It has become a main playground and research object for many serious and not so serious applications and models.

The quest to find the modeling language that I have in mind did not end till now. This paper collects some ideas and requirements that I came around on my career as a teacher, researcher and consulter for real problems and that I consider to be fundamental. It might stimulate persons with more competence in formal language design than I have to pick these ideas and to do a better job than I did till now. Although this paper is rather descriptive than formal, I am convinced that these ideas are worthwhile to be written down. Future will show of whether they are falling on fruitful soil.

9.1. Introduction

We use programming languages to formulate problems in order to deal with them by computers. At the same time, we want these languages to be readable for human beings. In fact, the emergence of computers has created a systematic way in which problems are formulated as *computations*. We now frequently make use of this mould even if computers are not involved. *Problem formulation as a computation* has a long tradition. The ancient Babylonians (19th–6th century B.C.), who had the most advanced mathematics in the Middle East, formulated all their mathematical problems as computations. They easily handled quadratic and even some cubic equations. They knew many primitive Pythagorean triples. They mastered the calculation of the volume of the truncated pyramid and many more impressive computations [25, 33]. Arithmetics was highly developed to ensure their calculations for architec-

tural and astronomical purposes. On the other hand, we cannot find a single instance of what we nowadays call a proof using symbolic variables and constraints. Only the *process of the calculation* – today we may say the *procedure* or the *algorithm* – was described. In this sense, the Babylonian mathematicians were in fact computer scientists. To illustrate how they did mathematics, one can consult the examples in [14]. This is in contrast with the Greek mathematicians who were interested in *why things are true* and less *how things are calculated*. The Babylonian and Greek approaches of acquiring and writing down knowledge are two fundamental different paradigms of knowledge: algorithmic knowledge and mathematical knowledge¹. This is the subject of this paper and how to represent the knowledge in programming or modeling languages.

9.2. Algorithmic Knowledge

As already mentioned, algorithmic knowledge has been used already by the Babylonians presented as written texts. We use algorithmic knowledge in our daily life in many situations, for instance, when we are preparing a recipe to cook spaghetti :

```
function CookSpaghetti() {
    Peel_Onions()
    ...   // various other tasks
    while (not Spaghetti_are_aldende) do Continue_cooking
        ()
    ...
    // finally
    repeat
        Serve_a_plate()
    until all_plates_have_been_served
}
```

In computer science we use programming languages to implement algorithmic knowledge in order to be executed by a computer. Various language paradigms have been developed since the first computer appeared: imperative, functional, and logic programming.

The **imperative programming paradigm** is closely related to the physical way of how (the von Neumann) computer works: Given a set of memory locations, a program is a sequence of well defined instructions on retrieving, storing and transforming the content of these locations. At each moment, the state of the computation can be represented by the momentary content of the memory

¹ When I use “mathematical” in the context of this text, I mean mainly applied mathematics used in operations research to solve problems.

locations (its memory-variables), instruction pointer and other register contents. At each step of the computation this state changes until the desired result (the goal state) is obtained, where the computation stops. The explicit notation of a sequential execution, the use of memory-variables (symbolic names) representing memory locations, and the use of assignment to change the values of variables are the characteristic building block of each imperative language. The imperative programming approach itself can be split into various sub-paradigms: (1) “spaghetti coding”, (2) structured-procedural, and (3) object oriented programming.

(1) *“spaghetti coding”* consists of a linear list of instructions (eventually containing goto’s (jumps) and if’s, for branching). The recipe above is a typical example. The code is not structured. Programming languages like Assembler or Basic favoured such an approach. In large software project today this approach is difficult to maintain. Assembler may be used still within other languages when a code must run efficiently, but in the light of better compilers this approach is becoming marginal. Higher level languages, like Basic, have almost completely vanished.

(2) *Structured and procedural programming* emphasizes the need that for larger projects one should break down the whole code into several more or less independent units, modules, packages, records, procedures and functions. This makes the code more maintainable and secure. The languages C and Pascal are prominent advocates of this approach. These languages also introduced the *concept of type*: Each variable is (implicitly or explicitly) attached to a type which is a defined data-structure template in memory. A type like “int” in C, for instance, consists of a given memory state and size (typically 4-bytes) together with a set of operations on its data (addition, subtraction, etc.). Additional data-structures can be declared by the programmer using *records* or *struct* and loosely coupled functions are implemented to manipulate these structures.

(3) *Object-oriented programming*, extends the concept of type: The user (programmer) can add new “types” called “classes” which consists of fields and tightly coupled methods (functions). Like in types, the fields are the memory state and the methods are the operations. The classes can be instantiated as objects, the correspondents to variables. Classes have many interesting features: Encapsulation structure and methods (hide them from outside), polymorphism and overriding (use the same procedure name for different types), hierarchy of classes, etc. Important representatives are languages Java, C++, Object Pascal.

Functional programming is in a sense opposite to imperative programming, where no states exist. The computation in functional programming is based on the evaluation of functions. Every program (i.e. computation) can be viewed as a function which translates an input into a unique output. Hence, a program can be represented by the function $I \xrightarrow{f} O$, I being the domain of

input, O being the domain of output, and f being the computation. A distinction is made between function definition, which describes how a function is to be computed, and function application, which is a call to a defined function. Since every value can be expressed as a function, there is no need to use explicitly variables, i.e. symbolic names for memory locations, nor do we need the assignment operation. Loops in the imperative programming languages are replaced by (tail) recursion. Furthermore, functions are first-class values, that is, they must be viewed as values themselves, which can be computed by other functions. The computational model of functional programming is based on the λ -calculus invented by Church A. (1936) as a mathematical formalism for expressing the concept of a computation; this was at the same time as Turing invented the Turing machine for the same purpose. One can show that the two formalism are equivalent in the strict sense that both can compute the same set of functions – called the computable functions. We call a programming language *Turing complete*, if it can compute all computable functions. With respect to functional programming we have the important result that: “A programming language is Turing complete if it has integer values, arithmetic functions on these values, and if it has a mechanism for defining new functions using existing functions, selection, and recursion.” [39, p. 12].

Logic programming emerged in the 1960s, when programs were written that construct proofs of mathematical theorems using the axioms of logic. This led to the insight that a proof can be seen as a kind of computation. But it was realised soon that also the reverse is true: computation can also be viewed as a kind of proof. This inspired the development of the programming language Prolog. In Prolog, a program consists of a set of (Horn)-rules together with at least one goal. Using the backtracking algorithm which consists of resolution and unification, the goal is derived from the rules. A logic programming language can be defined as a “notational system for writing logical statements together with specified algorithms for implementing inference rules.” [39, p. 426]. The unification-resolution algorithm is quite limited and has been replaced by various constraint solving mechanisms in the *constraint logic programming*, a modern extension of the logic programming class [36].

The functional and logic programming classes are sometimes called *declarative* in computer science. There are problem classes for which a declarative representation *can be used* directly to compute a solution, if interpreted in a certain way. A famous example is the algorithm of Euclid to find the greatest common divisor (gcd) of two integers. One can easily show that :

$$\text{gcd}(a, b) \equiv \begin{cases} \text{gcd}(b, a \bmod b), & \text{if } b > 0 \\ a, & \text{if } b = 0 \end{cases}$$

This is clearly a declarative way, since it *defines* the gcd. However, the formulation can directly be used to implement a functional (recursive) program

of finding the gcd, when the logical equivalence \equiv is interpreted as "can be substituted by". The underlying mechanism in a functional programming language is: "substitute the head of a function by its body recursively and simplify". The interpretation is essential, because it alone allows the underlying algorithm to be applied to the definition. Coded in a functional language (here *Scheme*) the algorithmic formulation of Euclid's algorithm is as following:

```
(define (gcd a b)
  (if (= b 0) a
      (gcd b (remainder a b))))
```

This paradigm of recursion is extremely powerful for designing algorithms. Hoare [27] wrote that he had no easy way to explain his Quicksort before he was aware of the recursion in Algol 60.

All problems, for which the instances can be reduced to smaller instances until finding a trivial (non-reducible) instance, can be treated in this way. This class of problem is surprisingly large and sometimes even more efficient algorithms can be discovered using this method. There are textbooks in algorithmic entirely based on this paradigm, an example is [79].

The case is similar in logic programming, which also is called declarative. In mathematics, the square function can be defined (declaratively) as:

$$\text{SQR}(x, y) \stackrel{\text{def}}{=} x = y^2$$

In Prolog the rule can be implemented as :

```
Sqr(X, Y) :- Y = X*X.
```

The query

```
? :- Sqr(X, 7).
```

will eventually return the answer $\{X = 49\}$, since it is easy to multiply $Y * Y$ ($7 * 7$) and to bind X to 49. If however the query

```
? :- Sqr(49, Y).
```

is made then the interpreter may not return a result, because it does not know what the inverse operation of squaring is. In a constraint logic language, the interpreter may or may not return a result, depending on whether the square root is implemented as an inverse of the square operation. This is but a simple example, but it shows an essential point: the logic program is declarative only in one direction, but not in the other, except when implementing *explicitly* the inverse into the system. This situation is similar in functional programming. The function (in Scheme)

```
(define (Sqr y)
  (* y y))
```

solves the problem: “given y find ($Sqry$)”. The inverse problem, however, which is implicitly also part of the mathematical definition, would be: “given the body $(*yy)$, which evaluates to a single value, find the value of y ”. This second problem cannot be solved using the functional definition. To find the square root, one needs an entirely different algorithm, e.g. Newton’s approximation formula:

$$x_n = \frac{x_{n-1} + \frac{a}{x_{n-1}}}{2}$$

In an algebraic modeling language (see below), such as LPL one can write:

```
|   variable x; y; constraint R: y*y = x; y=7;
```

or

```
|   variable x; y; constraint R: y*y = x; x=49;
```

The problem specification is valid in both directions and does not change, regardless of what are known and what are unknown quantities. By the way, a clever modeling language (which is not the case for *LPL*) would reduce the first model immediately to $\{y = 7, x = 49\}$ using only local propagation; the second model would be fed to a non-linear solver which eventually applies a gradient search method, for which Newton’s approximation is a special case.

The main point here is, that an algorithmic formulation of a problem assumes an (explicit or implicit) underlying mechanism (algorithm) for solving the problem, while a mathematical formulation (see below) does not assume such a mechanism. In this sense, problems formulated in either a functional or a logic programming language represent the problem in an algorithmic way, while algebraic modeling language *only* formulate the problem in a mathematical way, without indicating how to solve it.

This last example discloses a whole other class of problems that can also be formulated in a declarative way: the mathematical models. A more complete

example is a *linear program*², which can be represented declaratively in the following way:

$$\{\min cx \mid Ax \geq b\}$$

From this formulation – in contrast to the recursive declarative definitions above – nothing can be deduced that would be useful in solving the problem. However, there exists well-known methods, e.g. the Simplex method, which solves almost all instances in a very efficient way. Hence, to make the declarative formulation of a linear program useful for solving it, one needs to translate it into a form the Simplex algorithm accepts as input. A commercial standard formulation is the MPSX format [32] (see also [Wikipedia](#)). The translation from the declarative formulation $\{\min cx \mid Ax \geq b\}$ to the MPSX-form can be automated and is an essential task of the algebraic languages (see below).

All three programming paradigms (imperative, functional, and logic programming) concentrate on problem representation as a *computation*. Modern programming languages, like Python, JavaScript, etc., are typically multi-paradigmatic. The computation on how to solve the problem *is* the representation. We may call a notational system, that represent a problem by writing down its computation to find a solution, an *algorithmic language*. In this sense, all (three) presented programming classes consist of algorithmic languages.

Definition: An *algorithmic language* describes (explicitly or implicitly) the process (the computation) of solving a problem, that is, *how* a problem can be processed using a machine. The computation consists of a sequence of instructions which can be executed in a finite time by a Turing machine. The information of a problem which is captured by an algorithmic language is called *algorithmic knowledge* of the problem.

Algorithmic knowledge to describe a problem is so common in our everyday life that one may ask whether the human brain is “predisposed” to preferably look at a problem in describing its solution recipe (computation). Are we in fact predominantly using “algorithmic thinking”? (see also [15]).

² In the research community of operations research, a linear system with inequalities is called linear *program*. Dantzig wrote in his Linear Programming book [22]: “When I first analyzed the Air Force planning problem and saw that it could be formulated as a system of linear inequalities, I called my paper Programming in a Linear Structure. Note that the term ‘program’ was used for linear programs long before it was used as the set of instructions used by a computer. In the early days, these instructions were called codes.”

9.3. Mathematical Knowledge

Formulating a problem by writing down its computation for solving the problem is one way of representing it and many problems are preferably formulated this way – we'll see later why. It is the *algorithmic knowledge* of the problem solution that is described, it is the Babylonian way of problem representation. There exists another way to capture knowledge about a problem – it was already mentioned (example Linear Programming, above); it is the way to define *what* the problem is, rather than saying *how* to solve it. How can we say what the problem is? By defining the properties of the problem. Mathematically, the properties can be described as a feasible space which is a subset of a well-defined state space. Let X be a continuous or discrete (or mixed) state space ($\mathbb{R}^m \cup \mathbb{N}^n$) and let x be an arbitrary element within X ($x \in X$), then a mapping $R : X \rightarrow \{\text{false, true}\}$ defines a subset Y of X . This is often written as $Y = \{x \in X \mid R(x)\}$. Mathematical and logical formula can be used to specify the mapping R . These formula are called *constraints*. The unknown quantities x in $\{x \mid R(x)\}$ are called *variables*. The expression $Y = \{x \in X \mid R(x)\}$ is also called a *mathematical model* for the problem at hand. A notational system that represents a problem by writing down its properties using a state space is called a *mathematical language*.

Definition: A *mathematical language* describes the problem as a set of properties that can be expressed as a subset of a given state space. This space can be finite or infinite, countable or non-countable. The information of a problem which is captured by a mathematical language is called *mathematical knowledge* of the problem.

A mathematical representation of a problem is extremely powerful and applicable for a very wide range of problems. A constraint expression can be built of all kinds of mathematical and logical operations and functions to cover linear, non-linear, discrete or Boolean expressions. One can extend this notation to include special constraints (SOS), global constraints (a concept from constraint programming) (for example *AllDifferent*), or special variables (for example permutation variables, see my paper on permutation problems [71]). A complex problem can often be formulated in a very concise and readable way. Concise writings favour the insight to a problem: we can look at the formulation and grasp the essential “at a glance”. The following statement

$$\frac{z}{e^z - 1} = \sum_{n \geq 0} B_n \frac{z^n}{n!}$$

for example, gives us a concise definition of what the Bernoulli numbers B_n are [23, p. 285], namely the coefficients of this power series, in a way that we almost can “see” them, without knowing their actual values.

Nevertheless, a mathematical model has a big downside: it does not give any indication on how to solve³ it with the exception of fully enumerate the whole space and checking for each x whether $R(x)$ is true or false, but enumerating the space X is out of question for any realistic problem. There does not exist a unique method (algorithm) to solve all mathematical models. Even worse: Seemingly harmless problem can be arbitrarily difficult to solve. The problem of finding the smallest x and y such that $\{x, y \in \mathbb{N} \mid x^2 = 991y^2 + 1\}$, is a hard problem, because the smallest numbers contain 30 and 29 digits and apparently no other procedure than enumeration is known to find them. There are even problems that can be represented in a mathematical way, but cannot be computed. The problem of enumerate all even integers $\{x \in \mathbb{N} \mid 2x \in \mathbb{N}\}$, for example, cannot be computed, because the computation would take an infinite time. The problem of printing all real number within a unit circle $\{x, y \in \mathbb{R} \mid x^2 + y^2 \leq 1\}$ is even not enumerable. It is well-known, to give a nontrivial example, that the set of theorems in first-order logic, a restricted subset of classical logic, is undecidable (not computable). This can be shown by reducing the halting problem to the problem of deciding which logical statements are theorems [18, p. 520]. These examples simply have the shattering consequence: Given an arbitrary mathematical formulation of a problem, we can even not say in the general case whether the problem has a solution or not. The algorithmic way, on contrast, is a precise sequence of instructions (a recipe) of finding the solution to a given problem. For well designed algorithms, the sequence will terminate and we can even – for most algorithms – calculate an upper bound of its execution time – which is called its *complexity*.

So, why would we like to represent a problem mathematically, since we must solve it anyway and, hence, represent it finally as an algorithm? The reasons are *documentation* – besides *conciseness* and *insight*. In many scientific papers which deal with a mathematical problem and its solution, the problem is stated in a mathematical way for documental purposes. However, documentation is by no means limited to human beings. We can imagine an *mathematical language* implemented on a computer which is parsed and interpreted by a compiler. In this way, an interpretative system can *analyse the structure* of a mathematical program, can *pretty-print* it on a printer or a screen, can *classify* it, or *symbolically transform* it in order to view it as a diagram or in another textual form. There is a whole bunch of tasks we can execute using the computer for mathematical knowledge, besides solving the problem (see [64] for more details).

Of course, the burning question is, of whether the mathematical way of representing a problem could be of any help in *solving* the problem. The answer is *yes* and *no*. Clearly, the mathematical representation $\{x \in X \mid R(x)\}$, in its general form, does not give the slightest hint on how to find such a x . Problems represented in this way can be arbitrary complex to solve as seen before.

³ Solving a mathematical problem means to find one or all $x \in X$ such that $R(x)$ is true.

In practice, however, we are not so badly off. There exists problem classes, even with an infinite state space, which can be solved using general methods. An example is linear programming (as seen) and other classes of non-linear models. Great progress has been made in the last decades of discrete problems, more about them later on.

9.4. Historical Notes

Declarative and demonstrative mathematics originated in Greece during the Hellenic Age (as late as ca. 800–336 B.C.). The three centuries from about 600 (Thales) to 300 B.C. (*Elements of Euclid*) constitute a period of extraordinary achievement. In an economical context where free farmers and merchants traded and met in the most important marketplace of the time – the *Agora* in Athens – to exchange their products and ideas, it became increasingly natural to ask *why* some reasoning must hold and not just *how* to process knowledge. The reasons for this are simple enough: When free people interact and disagree on some subjects, then one asks the question of how to decide who is right. The answers must be justified on some grounds. This laid the foundations for a mathematical knowledge based on axioms and deductions. Despite this amazing triumph of Greek mathematics, it is not often realised that much of the symbolism of our elementary algebra is less than 400 years old [25, p. 179]. The concepts of the mathematical model as a declarative representation of a problem, are historically even more recent. According to Tarski, “the invention of variables constitutes a turning point in the history of mathematics.” Greek mathematicians used them rarely and in a very ad hoc way. Vieta (1540–1603) was the first who made use of variables on a systematic way. He introduced vowels to represent variables and consonants to represent known quantities (parameters) [25, p. 278]. “Only at the end of the 19th century, however, when the notion of a quantifier had become firmly established, was the role of variables in scientific language and especially in the formulation of mathematical theorems fully recognized” [1, p. 12].

It is interesting to note that this long history of mathematics with respect to algorithmic and mathematical representation of knowledge is mirrored in the short history of computer science and the development of programming languages. The very first attempt to devise an algorithmic language was undertaken in 1948 by K. Zuse [54]. Soon, FORTRAN became standard, and many algorithmic languages have been implemented since then.

The first step towards a declarative language was done by LISP. LISP is a declarative but not a mathematical language, since a function $I \xrightarrow{f} O$ can only be executed in one direction of calculation, from I to O. However, an important characteristic of a mathematical language is the symmetry of input and output. Prolog was one of the first attempts to make use of certain mathematical concepts, since a Prolog program is a set of rules. Unfortunately,

besides of other limitations, the order in which the rules are written is essential for their processing, showing that Prolog is not a mathematical language either. But Prolog had an invaluable influence on the recent development of *constraint logic programming* (CLP). Unfortunately, a constraint language normally come with its own solution methods based on constraint propagation and various consistency techniques and the modeling part and solution process are so closely intermingled that it is difficult to “plug in” different solvers into a CLP language. However, this is exactly what is needed. Since a general and efficient algorithm for all mathematical problems is unlikely, it is extremely important of being able to link a given mathematically stated problem to *different* solver procedure. The solution and formulation process should be separated as much as possible.

In logic programming, the underlying search mechanism (its computation) is behind the scene and the computation is intrinsically coupled with the language. This could be a strength because the programmer does not have to be aware of how the computation is taking place, one only writes the rules in a descriptive way and triggers the computation by a request. In reality, however, it is an important drawback, because, for most nontrivial problem, the programmer *must* be aware on how the computation is taking place. To guide the computation, the program is interspersed with additional rules (cut a.o.) which have nothing to do with the description of the original problem. This leads to programs that are difficult to survey and hard to debug and to maintain. This problem is somewhat relieved by the constraint logic programming paradigm, where specialized algorithms for different problem classes are directly implemented in the underlying search mechanism. However, this makes the language dependent of the implemented algorithms and problems that are not solvable by one of these algorithm cannot easily be stated in the language.

In the 80’s and 90’s of the last century, a new kind of mathematical languages, namely *algebraic modeling languages*, emerged in the community of operations research. The fact that a single method (for instance the Simplex method) can solve almost all linear programs, led to the development of these algebraic languages (such as GAMS, AMPL, LINGO, LPL a.o.), which have been becoming increasingly popular in the community of operations research. Algebraic modeling languages represent the problem in a purely mathematical way, although the different languages include also some computational facilities to manipulate the data. The mathematical notation itself of a model give no hint on how to solve the problem, it is stated in purely declarative way. Therefore, all algebraic modeling language must be linked to “solvers”, that is, mathematical libraries or programs that can solve the problem, for example, Gurobi, Cplex, XPress, HIGHS, etc. Many of these algebraic languages can also formulate non-linear problems, or some elements of constraint programming. They also include some methods to generate derivatives in order to link the model to non-linear solvers, like Knitro, IPOPT, etc. Typically,

an algebraic modeling language has an interface to many different (free and commercial) solvers.

One of their strength is the complete separation of the problem formulation as a mathematical model from the solution process. This allows the modelers not only to separate the two main tasks of model formulation and model solution, but also to switch easily between different solvers. This is an invaluable benefit for many difficult problems, since it is not uncommon that a model instance can be solved using one solver, and another instance is solvable only using another solver. Another advantage of some of these languages is to separate clearly between the mathematical *model structure*, which only contains parameters (place-holder for data) but no data, and the *model instance*, in which the parameters are replaced by a specific data set. This leads to a natural separation between model formulation and data gathering stored in databases and other data sources. Hence, the main features of these algebraic languages are:

1. Mathematical representation of the problem,
2. Clear separation between formulation and solution,
3. Clear separation between model structure and model data.

Algebraic modeling languages are limited in the sense that by definition no algorithm can be written using their notation. This is an important disadvantage and has led in the recent years to the advent of various packages in modern multi-paradigm programming languages. Especially in the Python community, several (mostly free) packages that allow to formulate mathematical models are increasingly popular at least in an academic context (for instance GurobiPy, Pyomo, PuLP, Python-MIP, APMonitor, a.o.). Also commercial companies are more and more switching to this approach of optimization problems. In some of these packages one can even switch easily between various solvers. The programming language Julia has a more unified approach with the package JuMP. Often the solver libraries offer themselves an interface to various programming languages (Python, C++, R, MatLab), to formulate the mathematical model (Gurobi, OR-tools, MiniZinc, CVXOPT, Knitro, etc.).

Indeed some commercial companies have developed their own mathematical modeling languages (AIMMS, MOSEL, HEXALY) that include some algorithmic features, but most of them are attached to their own exclusive solvers.

Let's recap the different programming language paradigms: Algorithmic knowledge can be implemented in various flavours and many programming languages have been created, they can be imperative or declarative. Most modern languages are multi-paradigmatic: In Python or JavaScript, for instance, one can program in a functional or in an object-oriented style.

algorithmic	spaghetti coding	imperative
	structured & procedural programming	
	object oriented programming	
	functional programming	
	logic programming	declarative
mathematical	constraint programming	
	algebraic programming	

Table 9.1: Programming Paradigms

Table 9.1 give an overview of the different programming paradigms.

9.5. Combine mathematical and algorithmic knowledge

Most problems can be formulated in several ways. Especially, a problem can be formulated using either algorithmic or declarative knowledge alone, or a combination of both. The question is of which is the most appropriate way. This depends on many criteria, such as efficiency, modifiability, extendability, readability, modularity, portability and others. It turns out that these criteria are exactly what in software engineering [3, 5, 13] is called *software quality criteria*. The main thesis of this paper is:

Thesis: With respect to these criteria, some problems are best formulated in a mathematical way, others in an algorithmic way, still others as a combination of both.

This thesis is now corroborated by some examples.

9.5.1 The Sorting Problem

Sorting an array of data is an important task in many contexts of data manipulation. Mathematically, the problem could be formulated as a permutation problem (see [71]). Let Π be the set of all permutations of some length n and let π be any permutation in Π . Let also π_i the i -th element within the permutation π , then the problem of (descending) sorting an array a_i could be formulated as an optimization problem (find a permutation $\pi \in \Pi$ such that $\sum_i^n i a_{\pi_i}$ is minimal):

$$\min_{\pi \in \Pi} \sum_i^n i a_{\pi_i}$$

This is a very elegant and short way to formulate the sorting problem, but is it a useful or efficient way? Not really, this formulation is in the same category as the TSP (traveling salesman problem) which can also be formulated as a permutation problem and which is NP-complete (find a round trip of minimal distances defined in a distance matrix $c_{i,j}$) :

$$\min_{\pi \in \Pi} \sum_i^n c_{\pi_i, \pi_j} \quad \text{where } j = i \mod n + 1$$

The question then is, how to find such a permutation. This is not an easy task. One could apply a general meta-heuristic, such as Tabu-search with a 2- or 3-opt neighbourhood to find nearly optimal permutations. However, this technique is only applicable for small instances and does not necessarily deliver an optimal solution. But why using this approach for sorting? We know that sorting is an “easy” problem: There exists efficient *algorithms* (Heapsort, Mergesort) that have a complexity of $O(n \log n)$. Even simple approaches such as Bubblesort can be implemented in a few lines of code in any programming language. So, while the mathematical approach seems elegant, it is neither practical nor efficient. The sorting problem is definitely better formulated as an algorithm.

9.5.2 The 0-1 Knapsack Problem

A 0-1 Knapsack problem is explained in [a knapsack model](#). Let $i \in I = \{1, \dots, n\}$ be a set of items. Each item i has a value w_i . We want to find a subset of items such the sum of values of its items is exactly K .

This problem can be formulated as a recursive function (dynamic programming!) as follows :

$$P(n, K) = \begin{cases} \text{true} & \text{if } n = 0, K = 0 \\ \text{false} & \text{if } n = 0, K > 0 \\ P(n - 1, K) \vee P(n - 1, K - w_n) & \text{otherwise} \end{cases}$$

Using memoizing⁴, this recursive function can be implemented with complexity of $O(nK)$.

The previous algorithmic approach is a perfect acceptable way to formulate the Knapsack problem. Probably a wider used approach is to formulate this problem as an integer linear mathematical model and using mixed-integer

⁴ Memoization is an optimization technique that stores computation results in cache, and retrieving that same information from the cache the next time it's needed instead of computing it again.

solvers (like Gurobi) to solve it. The formulation as a minimization problem is as follows :

$$\begin{aligned} \min \quad & \sum_{i \in I} x_i \\ \text{subject to} \quad & \sum_{i \in I} w_i x_i = K \\ & x_i \in \{0, 1\} \quad \text{forall } i \in I \end{aligned}$$

This mathematical formulation is straightforward: the variable x_i is 1 if the item i is in the subset, otherwise it is 0 and it is not in the subset. The constraint sums the values of all items in the subset and this must be K as required. In many contexts, the Knapsack problem contains additional constraints, therefore it is advantageous to formulate it as a mathematical model.

9.5.3 The Two-Persons Zero-Sum Game

The 2-persons zero-sum game is a classical problem from the game theory. There are two players, each with an associated set of strategies $i \in I = \{1, \dots, m\}$ and $j \in J = \{1, \dots, n\}$. While one player aims to maximize his payoff, the other player attempts to take an action to minimize this payoff. In fact, the gain of a player is the loss of another. Let $p_{i,j}$ be the payoff for the first player after his played strategy i and the second player responded with strategy j , and let x_i be the frequency in percent at which player one should apply strategy i , then these frequencies can be calculated by the following mathematical linear model (explanation are given in a concrete example in [Hofstadter's Game](#).

$$\begin{aligned} \max \quad & \min_j \left(\sum_i p_{i,j} x_i \right) \\ \text{subject to} \quad & \sum_i x_i = 1 \\ & x_i \geq 0 \quad i \in I = \{1, \dots, m\}, j \in J = \{1, \dots, n\} \end{aligned}$$

This is an efficient way to formulated the problem, because it could be transformed easily into an LP (linear program). It would be much more involving to formulate this problem in an algorithmic way.

9.5.4 The Cutting Stock Problem

The cutting stock problem is another classical problem from the operations research. It is the problem of cutting standard-sized pieces of stock material,

such as paper rolls or sheet metal, into smaller pieces of specified sizes while minimizing material wasted. The problem is explained in [cutstock](#). The column generation version implemented in LPL is found in [cutstock2](#). Technical details and a theory for these kind of problems can be found in the text book [80, Chap. 13]. To understand the main points that I want to make here, the technical details of the problem are not important.

My main point is the following: The cutting stock problem is preferably formulated partly in a mathematical way, partly in an algorithmic way:

For the mathematical part, three models must be implemented:

(1) A small instance of the cutting stock problem with only a few patterns as a relaxed linear model (we call it rCS) (details and the notation is found in the link above) :

$$\begin{aligned} \min \quad & \sum_j z_j \\ \text{subject to} \quad & \sum_j \text{patt}_{i,j} z_j \geq d_i \quad \text{forall } i \in I \quad (\text{A}) \\ & z_j \geq 0 \quad \text{forall } j \in J \end{aligned}$$

(2) An integer cutting stock (iCS) problem as follows :

$$\begin{aligned} \min \quad & \sum_j z_j \\ \text{subject to} \quad & \sum_j \text{patt}_{i,j} z_j \geq d_i \quad \text{forall } i \in I \quad (\text{A}) \\ & z_j \in \mathbb{Z}^+ \quad \text{forall } j \in J \end{aligned}$$

(3) Finally, a Knapsack problem (fP) to find new improving pattern is also used :

$$\begin{aligned} \min \quad & \sum_i d u_i y_i \\ \text{subject to} \quad & \sum_i w_i y_i \leq W \\ & y_i \in \mathbb{Z}^+ \quad \text{forall } i \in I \end{aligned}$$

For the algorithmic part, a procedure (in pseudo-code) is implemented to solve these mathematical models several times :

```

function ColumnGenertion() {
    InitializePatterns() //add some patterns to Cutstock
    solve Cutstock //returns duals
}

```

```

Inject_duals_into_Knapsack()
solve Knapsack      //to find a new improving pattern
while (found_an_improved_pattern) do {
    Add_the_pattern_to_Cutstock()
    solve Cutstock      //returns duals
    Inject_duals_into_Knapsack()
    solve Knapsack      //to find a new improving pattern
}
solve IntegerCutstock
}

```

9.5.5 The Traveling Salesman Problem (TSP)

There are many different model formulations of the TSP. One concise mathematical formulation is the so-called *Cutset formulation* (see [tsp-5](#)). It is as follows:

$$\begin{aligned}
 & \min \quad \sum_{i,j \in V} c_{i,j} x_{i,j} \\
 & \text{subject to} \quad \sum_{j \in V} x_{i,j} = 1 \quad \text{forall } i \in V \quad (A) \\
 & \quad \sum_{i \in V} x_{i,j} = 1 \quad \text{forall } j \in V \quad (B) \\
 & \quad \sum_{i \in S} \sum_{j \in S} x_{i,j} \leq |S| - 1 \quad \text{forall } S \subset V, |S| \geq 2 \quad (G) \\
 & \quad x_{i,j} \in \{0, 1\} \quad \text{forall } i, j \in V, i \neq j \\
 & \quad i, j \in V = \{1 \dots n\}, n > 0
 \end{aligned}$$

The binary variables $x_{i,j}$ is 1 if the circuit contains the arc (i, j) in the graph, otherwise it is 0. The constraints A and B make sure that a location is visited only once, and the constraint G excludes all subtours (further explications is given in [tsp-5](#). Unfortunately, G contains an exponential number of constraints, for each sub-tour one. Hence, it is not possible to generate all of them at once. What is possible, however, is to solve an problem only with the constraints A and B then check which subtours constraints are violated and add them to the model, and proceed this way until no subtours is violated. The (algorithmic) procedure is then to begin with an empty list S and repeatedly add subtours to S :

```

function AddSubtours() {
    S = {}
    cNr1 = 2    // initialize cNr1 (number of components)
}

```

```

while (cNr1 > 1) do
    solve Tsp
    cNr1 = Nr_Components_Of_Graph(C) //get the
        components C of the graph
    Add_the_Components_to_S(C)
}
}

```

9.6. Requirements

Combining mathematical and algorithmic knowledge in a single language is a great advantage for formulating such problems and the combination should be in a way that the different components can be clearly separated from each other. We shall call a modular language that combines mathematical and algorithmic knowledge, *modeling language*.

Definition: A *modeling language* is a notational system which allows us to combine in a modular way mathematical and algorithmic knowledge in the same framework. The content captured by such a notation is called *model*.

Algebraic modeling languages capture only mathematical knowledge, programming language capture only algorithmic knowledge. In my opinion, we are still missing a well designed modeling language. My own attempt is the language LPL. In LPL, one can implement models in a modular way, the language is close to the mathematical notation for mathematical models, it is also Turing complete. It is great and efficient to formulate large MIP models. However, it lacks classes and other advanced data structures, it lacks real (recursive) stack-based functions, and the algorithmic part is interpreted and might be slow. Although the basic language is simple and straightforward, some language constructs are difficult to understand, because they have grown over time.

Well, one could argue that modern programming languages, such as Python and Julia, contain already all elements that we need to implement (algorithmic+mathematical) models. Several packages have been implemented – as also mentioned – and new packages are implemented all the time to formulate mathematical models, like JuMP (Julia), GurobiPy, Pyomo, PuLP, Python-MIP, pyOPT (Python).⁵ Most solver libraries have an interface to Python or other programming languages. This approach is actually very trendy. In a

⁵ A recent list of optimization packages in Python: Update 2024 can be found at [Python package 2024](#).

concrete context, this approach may be the appropriate way, but in general, I think, this is a short-term thinking.

It is said that this approach has some advantages: One must learn only one language; a language like Python contains thousands of packages for all kinds of tasks: manipulating, reading/writing data, generating graphics and others; implementing efficient algorithms. So then, if one wants to use a “master language” like Python for modeling, why not using an algebraic language as an additional package (library), instead of using complicated syntax to define mathematical variables and constraints within the programming language?⁶

As a small example, let's look at the following simple non-linear model (see [gurobiNLP](#)):

$$\begin{aligned} \min \quad & \sin(x) + \cos(2x) + 1 \\ \text{subject to} \quad & 0.25e^x - x \leq 0 \\ & -1 \leq x \leq 4 \end{aligned}$$

Why should one use the following GurobiPy (Python) code to implement this model

```
import gurobipy as gp
from gurobipy import GRB

m = gp.Model()
x = m.addVar(lb=-1, ub=4, vtype=GRB.INTEGER, name="x")
twox = m.addVar(lb=-2, ub=8, name="2x")
sinx = m.addVar(lb=-1, ub=1, name="sinx")
cos2x = m.addVar(lb=-1, ub=1, name="cos2x")
expx = m.addVar(name="expx")
m.setObjective(sinx + cos2x + 1, GRB.MINIMIZE)
lc1 = m.addConstr(0.25 * expx - x <= 0)
lc2 = m.addConstr(2.0 * x - twox == 0)
gc1 = m.addGenConstrSin(x, sinx, "gc1")
gc2 = m.addGenConstrCos(twox, cos2x, "gc2")
gc3 = m.addGenConstrExp(x, expx, "gc3")

m.params.FuncNonlinear = 0      ## using Piecewise linear
approx
# m.params.FuncNonlinear = 1    ## using MINLP solver
# m_presolve = m.presolve()
m.optimize()
```

if one can implement this model as follows (in LPL):

⁶ The package [amplpy](#) goes in this direction. LPL also could be called from Python. LPL can be used as a DLL (dynamic link library) and linked to any language.

```

model gurobiNLP;
  SetSolver(gurobiLSol);
  variable x [-1..4];
  constraint A: 0.25*Exp(x) - x <= 0;
  minimize   M: Sin(x) + Cos(2*x) + 1;
end

```

It should be the modeling language's task to decompose complicated non-linear terms to feed them to Gurobi (that's what LPL does automatically). A further big advantage of the LPL formulation is that the model can easily be sent to another solver like Knitro (or any other solver that has an interface to LPL) just by replacing the first instruction to

```
| SetSolver(knitroLSol);
```

In this case, LPL does not need to decompose the non-linear expressions, but it eventually generate derivatives instead.

Anyway in my opinion, mathematical modeling should not be just an "addon" package within a common programming language, this would always be somewhat artificial. The mathematical structure is hidden within the programming code and the mathematical notation must be bent into shape of the particular programming language. Modeling is too important to be a supplement or an annex of an (existing) programming language. We need a proper (sub-)language for the mathematical part. The modeling language that I have in mind (which has not been implemented by now in my opinion) is also a complete programming language with all its interfaces to libraries, other software and own extensions with packages.

In the following, I list my personal preferences in the light of the following criteria for a – what I think is a "good" – mathematical modeling system. A modeling system should fulfill the following requirements:

1. *A modeling system should be based in a formal language.* Like a programming language, which is based on a written text and a formal grammar, a modeling system should be based on an executable language.
2. *The modeling language should be a complete algorithmic and mathematical language,* that is, rich enough to implement any algorithm (Turing complete), and it should contain features of a modern programming language, such as functions as a first class entity, classes, etc. It should also contain explicit syntax to formulate mathematical models.
3. In the language, it should be possible to *modularize an model into sub-modules*, in order to encapsulate entities and objects, similar to a modern programming language that contains classes (objects), modules, procedures, functions, units, etc.

4. The mathematical part of the modeling language syntax should be as close as possible to a common mathematical notation used to specify a model.
5. An important part of the syntax should be its (*sparse*) *index capability* in order to be able to formulate large models in a concise way.
6. It should be possible to formulate within the language *all kinds of model paradigms*: linear, non-linear, permutations, containing logical constraints, constraint programming (CP), differential systems, etc.
7. *Reformulation and transformation* of a mathematical model should be integrated. Depending on the solver used, a model must eventually be reformulated: Using a MIP solver, for instance, needs all constraints to be linear and a Boolean expression must be translated into mathematical expressions. Non-linear expressions must be possibly decomposed (as see before). Global constraints in CP are to be translated into mathematical constructs. etc.
8. The model formulation should be *independent of a solution method* (solver). From the structure of the model, it should be possible to infer automatically what solver is apt to solve the mathematical model. The modeler can overrule the assignment by various methods.
9. The mathematical model structure should be *independent from the data instances*. A model structure should be executable without data. The data could be part of the code, but it is not necessary and normally would be separated from the model structure. Data are best stored in databases or other external files.
10. On the base of the language, *visual representations*, like A-C graphs or others, and “visual” editors to build, manipulate and to modify the model could be built, as alternatives to editing and viewing the textual code – a modeling framework, as we know it from many programming framework (for example RAD Studio from Embarcadero).
11. Likewise programming languages, *documenting a model* is an important part in modeling – I guess it is even more important in modeling to understand its semantic.

9.7. Conclusion

In this paper, a new type of modeling language has been presented which is a superset of a programming language, since it can not only represent algorithmic, but also mathematical knowledge. It was shown by examples that some problems are best formulated in a mathematical way, others in an algorithmic way, still others in a combination of both. Consequently, we need languages which can code both kinds of knowledge in a unified framework.

Many researchers, especially in operations research, think that a modeling language is a specialized (programming) language. The concept of *modeling language* is often identified with that of an *algebraic language* and extending it with some algorithmic concepts is enough. I have tried to convince the reader that a modeling language should be more – no less – than a programming language. In fact, the activity of modeling is more general than that of programming. Programming is just a particular activity of modeling. Thus, a program can be viewed as a special kind of a model. Consequently, a modeling language must subsume a programming language, and must be more general.

Modeling, that is, the skill to translate a problem into a mathematical and algorithmic formalism (or some coding), should no longer be an “ad hoc science” where “everything goes”. It should be – like programming, the skill to write algorithms – an art to write *good* models. Furthermore, modeling should follow certain criteria of quality in the same way as software engineering teaches it for writing algorithmic programs.

The main criteria are: *reliability* and *transparency*. Reliability can be achieved by a unique notation to code models, and by various checking mechanisms (type checking, unit checking, data integrity checking and others). Transparency can be obtained by flexible decomposition techniques, like modular structure as well as access and protection mechanisms. Hence, what is true for programming languages is even more true for modeling languages. The cutting stock problem show clearly, that we need languages, which follow these criteria.

In the paper [8] the authors argues that the 5-th generation of programming languages will be modelware, i.e. programming will be replaced by modeling. We shall see.

BIBLIOGRAPHY

- [1] Tarski A. *Introduction to Logic and to the Methodology of the Deductive Sciences*. Oxford University Press, 1994, 4th edition.
- [2] Fox W.P. Albright B. *Mathematical Modeling with Excel*. CRC Press, New York, 2020, second edition.
- [3] MEYER B. *Object–Oriented Software Construction*. Prentice Hall, London, 1997.
- [4] Shetty C.M. Bazaraa M.S., Sherali H.D. *Nonlinear Programming, Theory and Algorithms*. Wiley, 2006, third edition.
- [5] Pogh J. Bell D., Morrey I. *The Essence of Program Design*. Prentice Hall, London, 1997.
- [6] Weismantel R. Bertsimas D. *Optimization over Integers*. Dynamic Ideas, Belmont, 2005.
- [7] Hart W. E. Laird C. D. Nicholson B. L. Siroila J. D. Watson J-P Woodruff D. L. Bynum M. L., Hackebel G. A. *Pyomo–optimization modeling in python*, volume 67. Springer Science & Business Media, third edition, 2021.
- [8] Mancas C. On Modelware as the 5th Generation of Programming Languages. *Acta Scientific Computer Sciences*, Vol 2 Issue 9, 2020.
- [9] Pedregal P. García R. Alguacil N. Castillo E., Conejo A.J. *Building and Solving Mathematical Programming Models in Engineering and Science*. Wiley, 2002.
- [10] Kwon Ch. *julia – programming for operations research*. <http://www.chkwon.net/julia>, second edition, 2019.

- [11] Hooker J.N. Chandru V. *Optimization Methods for Logical Inference*. J. Wiley Sons, Inc., 1999.
- [12] Marinoni M. Cilloni A. Spreadsheet, Chessboard and Matrix Accounting: The origin and development of advanced accounting instruments. *The 10th World Congress of accounting Educators*, 2006, Istambul.
- [13] Farley D. *Modern Software Engineering: Doing What Works to Build Better Software Faster*. Addison-Wesley Professional, 2021.
- [14] Knuth D.E. *Selected Papers on Computer Science*. Cambridge University Press. Chapter 11, Ancient Babylonian Algorithms, (first print 1972), 1996.
- [15] Knuth D.E. *Selected Papers on Computer Science*. Cambridge University Press. Chapter 4, Algorithms in Modern Mathematics and Computer Science, (first print 1972), 1996.
- [16] More J.J. Dolan E.D. Benchmarking Optimization Software with COPS,. Tech. Report. Mathematics and Computer Science Division,, 2000.
- [17] Hedengren JD. et al. Nonlinear modeling, estimation and predictive control in apmonitor. *Chemical Engineering*, 2014.
- [18] Beigel R. Floyd R. W. *The Language of Machines, An Introduction to Computability and Formal Languages*. Computer Science Press, 1994.
- [19] Kernighan B.W. Fourer R., Gay D.M. *AMPL: A Modelling Language for Mathematical Programming*. Duxbury Press/Brooks/Cole Publ. Co., second edition, 2003.
- [20] Polya G. *How To Solve It*. Penguin Books 1990, London, 1957.
- [21] Polya G. *Mathematical Discovery, On Understanding, Learning, and Teaching Problem Solving*. John Wiley, Combined Edition, 1968.
- [22] Dantzig G.B. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.
- [23] Patashnik O. Graham R.L., Knuth D.E. *Concrete Mathematics*. Addison-Wesley Publ. Comp, second edition, 1994.
- [24] Schneider F.B. Gries D. *A Logical Approach to Discrete Math*. Springer, p 37, 1994.
- [25] Eves H. *An Introduction to the History of Mathematics, sixth edition*. The Saunders Series, Fort Worth, 1992.
- [26] Dudeney H.E. *Amusements in Mathematics*. Thomas Nelson and Sons, 1917.
- [27] Jones C.B. Hoare C.A.R. *Essays in Computing Science*. Prentice Hall, New York, 1989.

- [28] Williams H.P. Logical problems and integer programming. *Bull. Inst. Math. Appl.*, 13:18–20, 1977.
- [29] Williams H.P. An alternative explanation of disjunctive formulations. *European Journal of Operations Research*, 72:200–203, 1994.
- [30] Williams H.P. *Logic and Integer Programming*. Springer, 2009.
- [31] Williams H.P. *Model Building in Mathematical Programming*. John Wiley, Fifth Edition, 2013.
- [32] IBM. *Mathematical Programming System Extended/370 (MPSX/370), Version 2, Program Reference Manual*. IBM, 1988.
- [33] Yaglom I.M. *Mathematical Structures and Mathematical Modelling*. Gordon and Breach Science Publ., New York, (transl. from the Russian by D. Hance, original ed. 1980), 1986.
- [34] Lucas H.C.J.R. Isakowitz T., Schocken S.
- [35] Schwichtenberg J. *Teach Yourself Physics, a travel companion*. No-Nonsense Books, 2020.
- [36] Maher M.J. Jaffar J. *Constraint logic programming: a survey*. The Journal of Logic Programming, Volumes 19–20, Supplement 1, May–July 1994, Pages 503–581, 1996.
- [37] Hooker J.N. A Quantitative Approach to Logical Inference. *Workshop "Mathematics and AI", Vol II, FAW Ulm, 19th-22nd December 1988*, pages 289–314, 1988.
- [38] Malliaris M. Jukic N., Jukic B. Processing (OLAP) for Decision Support. *Handbook on Decision Support Systems 1*, eds. Burstein F., Holsapple C.W., Springer, pp. 259ff, 2008.
- [39] Louden K.C. *Programming Languages – Principles and Practice*. PWS-KENT Publ. Comp., 1993.
- [40] Cesari L. *Optimization – Theory and Applications*. Springer Verlag, 1983.
- [41] Schrage L. Optimization modeling with LINGO. <http://www.lindo.com>, 1999.
- [42] Wolsey L.A. *Integer Programming*. Wiley, 1998.
- [43] Levitin Maria Levitin A. *Algorithmic Puzzles*. Oxford University Press, 2011.
- [44] Biegler L.T. *Nonlinear Programming, Concepts, Algorithms, and Applications to Chemical Processes*. MOS-SIAM Series on Optimization, 2010.
- [45] Wallace M. *Building Decision Support Systems, using MiniZinc*. Springer, 2020.

- [46] Williams McKinnon. 1989.
- [47] Düsing R. Meier G. Zur Modellierung logischer Aussagen ergänzend zu Linearen Programmen, Grundlagen und Entwurfsüberlegungen für einen Modellgenerator. *OR-Spektrum*, Vol 14, p. 149–160, 1992.
- [48] Arnott D. Meredith R., O'Donnell P. Databases and Data Warehouses for Decision Support. *Handbook on Decision Support Systems 1*, eds. Burstein F., Holsapple C.W., Springer, pp. 207ff, 2008.
- [49] Moody S. Mitra G., Lucas C. Tools for reformulation logical forms into zero-one mixed integer programs. *European Journal of Operational Research*, 72:2:262–276, 1994.
- [50] Houle C. Murray P. You Can't Always Get from A to B by Way of X and Y. *a Quantrix White Paper at www.quantrix.com*, 2006.
- [51] Wolsey L.A. Nemhauser G.L. *Integer and Combinatorial Optimization*. Wiley, 1988.
- [52] Seel N.M. Model-based learning: a synthesis of theory and research. *Educational Technology Research and Development*, January, 2017.
- [53] Barth P. *Logic-Based 0-1 Constraint Programming*. Kluwer Academic Publishers, p.28, 1996.
- [54] NAUR P. *The European Side of the Last Phase of the Development of ALGOL 60*. in: History of Programming Languages, Wexelblat R.L. (ed.), (from the ACM SIGPLAN History of Programming Languages Conference, June 1–3, 1978), Academic Press., 1981.
- [55] Rivett P. *Model Building for Decision Analysis*. Wiley, 1980.
- [56] Wolsey L.A. Pochet Y. *Production Planning by Mixed Integer Programming*. Springer, 2006.
- [57] Bosch R. Mind Sharpener. OPTIMA MPS Newsletter, January 2000.
- [58] Mattessich R. A Concise History of Analytical Accounting: Examining the Use of Mathematical Notions in our Discipline. *De Computis Spanish Journal of Accounting History*, Vol. 2, pp 123ff, 2005.
- [59] Jeroslow R.G. *Logic-Based Decision Support, Mixed Integer Model Formulation, Annals of Discrete Mathematics* Vol 40. North-Holland, 1989.
- [60] Robinson S. Historical and Philosophical Perspectives on the Activity of Modelling. *Proceedings of the Operational Research Society Simulation Workshop 2023*, 2023.
- [61] Thall R.M. Seiford L.M. Recent developments in DEA: the mathematical programming approach to frontier analysis. *Journal of Econometrics*, Vol 46, p. 7–38, 1990.

- [62] Winston W.L. Seref M.H., Ahuja R.K. Developing Spreadsheet-Based Decision Support Systems (Using Excel and VBA for Excel). 2007, Belmont, MA: Dynamic Ideas.
- [63] Hürlimann T. An Introduction to the Modeling Language LPL. <https://matmod.ch/lpl/doc/modeling3.pdf>.
- [64] Hürlimann T. Computer-Based Mathematical Modeling (Habilitation Thesis). <https://matmod.ch/lpl/doc/WP-old/1997-Habil-new.pdf>.
- [65] Hürlimann T. Functionality Sheet of LPL. <https://matmod.ch/lpl/doc/lpl.pdf>.
- [66] Hürlimann T. How to model ? <https://matmod.ch/lpl/doc/modeling2.pdf>.
- [67] Hürlimann T. Index Notation in Mathematics and Modeling Language LPL: Theory and Exercises. <https://matmod.ch/lpl/doc/indexing.pdf>.
- [68] Hürlimann T. Logical modeling. <https://matmod.ch/lpl/doc/logical.pdf>.
- [69] Hürlimann T. LPL Tutorial, Models. <https://matmod.ch/lpl/doc/tutorial.pdf>.
- [70] Hürlimann T. My Vision of a Modeling Language. <https://matmod.ch/lpl/doc/modeling5.pdf>.
- [71] Hürlimann T. Permutation Problems. <https://matmod.ch/lpl/doc/permuation.pdf>.
- [72] Hürlimann T. Reference Manual for the LPL Modeling Language, most recent version. <https://matmod.ch/lpl/doc/manual.pdf>.
- [73] Hürlimann T. Various Model Types. <https://matmod.ch/lpl/doc/variants.pdf>.
- [74] Hürlimann T. Various Modeling Tools. <https://matmod.ch/lpl/doc/modeling4.pdf>.
- [75] Hürlimann T. What is modeling ? <https://matmod.ch/lpl/doc/modeling1.pdf>.
- [76] Hürlimann T. *Mathematical Modeling and Optimization – An Essay for the Design of Computer-Based Modeling Tools*. Kluwer Academic Publ., Dordrecht, habilitation at the university of fribourg edition, 1999.
- [77] Hürlimann T. Modeling Languages in Optimization: a New Paradigm. in : *Encyclopedia of Optimization*, eds. Floudas A., Pardalos P.M., Springer, 2001.

- [78] Kuhn T.S. *The Structure of Scientific Revolutions*. University of Chicago Press, 1962.
- [79] Manber U. *Introduction to Algorithmics, A Creative Approach*. Addison-Wesley Publ. Comp., Reading, 1989.
- [80] Chvátal V. *Linear Programming*. W.H. Freeman Company, New York, 1983.
- [81] Remigijus Paulavicius Vaidas Jusevicius, Richard Oberdieck. Experimental analysis of algebraic modelling languages for mathematical optimization. *Informatica*, 1–22, 2021.
- [82] Hong Y. Williams H.P. Representations of the all-different Predicate of Constraint Satisfaction in Integer Programming. *INFORMS Journal on Computing*, 13:96–103, 2001.
- [83] Winston W.L. *Operations Research, Applications and Algorithms*. Duxbury, 3rd ed., 1998.