Rapid Mathematical Programming

vorgelegt von Diplom-Mathematiker

T K

Flensburg

Fakultät II – Mathematik und Naturwissenschaften der Technischen Universität Berlin
zur Erlangung des akademischen Grades

D N

genehmigte Dissertation

Promotionsausschuß

Vorsitzender	Prof. Dr. Fredi Tröltzsch
Berichter/Gutachter	Prof. Dr. Martin Grötschel
Berichter/Gutachter	Prof. Dr. Alexander Martin

Tag der wissenschaftlichen Aussprache: . Dezember

Berlin
D

Rapid Mathematical Programming

Thorsten Koch

This book was typeset with TEX using LATEX and many further formatting packages.
The pictures were prepared using , , and .

All numerals in this text are recycled.

Für meine Eltern

Preface

Avoid reality at all costs
— fortune( )

As the inclined reader will find out soon enough, this thesis is not about deeply involved mathematics as a mean in itself, but about how to apply mathematics to solve real-world problems. We will show how to shape, forge, and yield our tool of choice to rapidly answer questions of concern to people outside the world of mathematics.

But there is more to it. Our tool of choice is software. This is not unusual, since it has become standard practice in science to use software as part of experiments and sometimes even for proofs. But in order to call an experiment scientific it must be reproducible. Is this the case?

Regarding software experiments, we have first to distinguish between reproducing and repeating results. The latter means that given the same data and the same programs, running on a similar computer, the results are identical. Reproducing results means to use the same data, but di erent programs and an arbitrary computer.

Today we can reproduce experiments conducted by Leonardo da Vinci in the fifteenth century. But can we expect even to repeat an experiment that involves a -year-old (commercial) program? Or in case we try to reproduce it, using di erent software and get dissenting results, can we make any conclusions why the results di er?

Software is getting more complex all the time. And by no means it is usually possible to prove the correctness of a specific implementation even if the algorithm employed is known to be correct. But what can we do if the source code of the program is not available at all? How could we assert what the program is really doing?

Science is about using the work of fellows. If a proof or experiment is published anybody can use it as part of their work, assumed due credit is given to the author. But how can we build on software that is not freely available?

The same questions can be asked for the input data. Without the precise data, computational experiments can neither be repeated nor reproduced. Surely part of the problem is that there is no accepted or even adequate way to publish software and data for review in the same way as articles are published. While this thesis by no means gives answers to the above questions[1], considerable e orts were taken to set a good example regarding the correctness of the software and to improve the possibilities for the reader to repeat or reproduce the results shown.

Acknowledgements

I am deeply indebted to all my friends and colleagues at , who supported me in writing this thesis. They showed me that teamwork does not mean to work in a crowd, but to improve the quality of the work by combining the knowledge of many people.

I would especially like to thank my supervisor Prof. Martin Grötschel, who started his work in Berlin just at the right time to attract me to combinatorics and optimization.

1 For more information on these topics, see Borwein and Bailey ( ), Greve ( ), Johnson ( ).

The environment and challenges he has provided me are the most fertile and stimulating I have encountered so far.

This thesis would not have happened without Prof. Alexander Martin, who recruited me two times and taught me much of what I know about optimization.

Many thanks go to my brave proof readers Monica Ahuna, Andreas Eisenblätter, Sven Krumke, Roland Wessäly and especially Tobias Achterberg, Volkmar Gronau, Sylwia Markwardt, and Tuomo Takkula.

And last but not least, I would like to thank Ines for her inexhaustible support and Keiken for inexhaustibly trying to distract me.

Berlin, October
Thorsten Koch

Contents

Chapter 1

Introduction

Ninety-Ninety Rule of Project Schedules:
The first ninety percent of the task takes ninety percent of the time,
and the last ten percent takes the other ninety percent.
— fortune( )

This thesis deals with the implementation and use of out-of-the-box tools in linear mixed-integer programming. It documents the conclusions drawn from five years of implementing, maintaining, extending, and using several computer codes to solve real-life problems.

Although most of the projects carried out were about location planning in the telecommunication industry, the lessons learned are not tied to a specific area. And while many of the topics presented might be well-known in general, we hope to provide some new insights about details of implementation, integration, and the difficulties that can arise with real-world data.

What makes general tools attractive?

In our experience customers from industry share a few attributes. They

- do not know exactly what they want,
- need it next week,
- have not collected yet the data necessary,
- usually have not really the data they need,
- often need only one shot studies,
- are convinced *"our problem is unique"*.

This mandates an approach that is fast and flexible. And that is what general tools are all about: Rapid prototyping of mathematical models, quick integration of data, and a fast way to check if it is getting to be feasible. Due to the ongoing advances in hardware and software, the number of problems that can be successfully tackled with this approach is steadily increasing.

1.1 Three steps to solve a problem

According to Schichl ( ) the complete process to solve a problem looks like that depicted in Figure . . Given availability of data[1] which represents instances of a problem we can summarize this to three tasks:

i) Build a mathematical model of the problem.
ii) Find or derive algorithms to solve the model.
iii) Implement the algorithms.

Of course this sounds much easier than it usually is. First we have to gain enough understanding of the problem to build a mathematical model that represents the problem well enough such that any solution to an instance of the model is meaningful to the real problem. Then we have to turn the theoretical model into an algorithm, i. e., we need a model that leads to (sub-) problems that can be solved in practice. And after that the algorithms have to be implemented e ciently. Very often something is lost in each stage:

- The model is not a perfect representation of the reality, i. e., some aspects had to be left out.
- The algorithms are not able to solve the model in general, e. g., the problem might be $\mathcal{NP}$-hard.
- Not the perfect algorithm was implemented, but only a heuristic.

From now on we assume that the model resulting from step one is an / / model. Under these assumptions many tools are available to make steps two and three easier. We will try to provide a short classification here: For problems e cient algorithms like the *barrier method* are available. Also *simplex* type algorithms have proven to be fast in practice for a long time. Both of these algorithms can, given enough memory and computing power, solve problems up to several million variables and side constraints.

Sometimes only an approximation of the optimal solution of a linear program is needed and algorithms like *Lagrange Relaxation* or *Dual Ascent* provide fast solutions.

In general, *Branch-and-Bound* type algorithms are the most successful ones to solve (mixed) integer programming problems. Depending on the problem there are many possibilities for upper and lower bounding. The most successful general technique by now is to solve the relaxation of the problem to gain lower bounds.

While many combinatorial problems like *shortest path*, or *min-cost flow* can in principle also be solved as a linear program, special algorithms are available that are often extremely fast in practice.

1 This is a rather strong assumption as it turned out in every single project we conducted that the preparation of the input data was an, if not *the*, major obstacle.

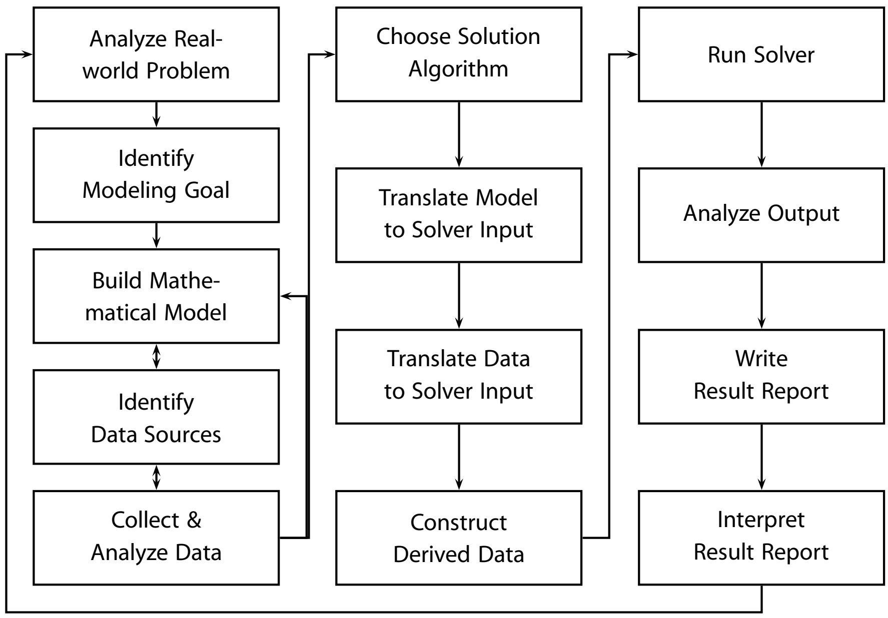


Figure . : Modeling cycle according to Schichl ( )

Now, how can we apply this vast toolbox? There are four principle ways to go, but of course transitions between these are smooth and all kinds of combinations are possible:

- Use a mathematical workbench like [2] or [3].
- Use a modeling language to convert the theoretical model to a computer usable representation and employ an out-of-the-box general solver to find solutions.
- Use a framework that already has many general algorithms available and only implement problem specific parts, e. g., separators or upper bounding.
- Develop everything yourself, maybe making use of libraries that provide high-performance implementations of specific algorithms.

If we look at Figure . we get a rough idea how to rate each method. The point marked *TP* is the level of e ect that is typically reached with each method, i. e., while it is possible to outperform the framework approach by do-it-yourself, this is usually not the case.

1.1.1 Mathematical workbench

This is the easiest approach, provided that the user is familiar with the workbench he wants to use. The time needed to get used to a specific workbench can be considerable. If this hurdle is overcome, everything needed is already built-in.

2 http://www.mathworks.com
3 http://www.wolfram.com

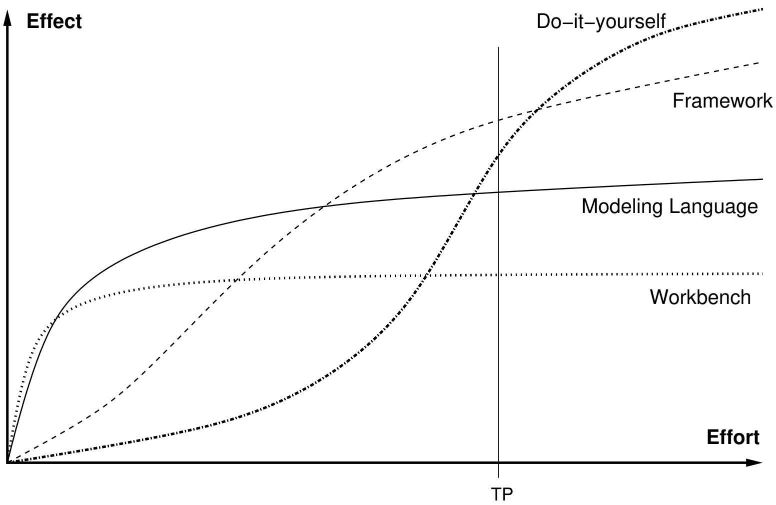


Figure . : “E ort vs. e ect”

On the other hand this is also the most restricted approach. The performance of the built-in solvers is usually not state-of-the-art. And if anything of the built-in functionality is missing or not good enough, not much can be done.

1.1.2 Modeling Language with out-of-the-box solver

This is the most flexible approach on the modeling side. It is very easy to get used to and provides nearly immediate results. No serious programming is required and it is very easy to test di erent modeling approaches. It is the only approach that allows easily to switch between di erent state-of-the-art solver engines.

On the downside we have considerable algorithmical restrictions regarding the solution process. No real separation of inequalities or problem specific upper bounding is possible. And if the out-of-the-box solver is not able to solve the problem, the model has to be changed to contrive an easier problem.

1.1.3 Framework

Frameworks are the most economic way to implement sophisticated high-performance problem solvers. Many high-performance building bricks are readily available and lots of general tasks are prebuilt. It is a very flexible approach and will lead for most projects to the best performing result in the end.

Unfortunately frameworks usually require the user to adopt the view of the world as seen from its designers, which can lead to some startup delay. And, what is even more important, today‘s frameworks are so vendor specific that it is nearly impossible to transfer an implementation built with one framework to another. If a framework was chosen, you have to stick to it. And if you need something the framework does not provide there might be no good way to get it. In general using self-written lower bounding algorithms might prove di cult with many frameworks.

1.1.4 Doing it all yourself

Nothing is impossible in this approach of course, except maybe a quick success. For most problems, this is tedious, error prone and requires the reimplementation of many well-known but sophisticated structures. Of course some problems benefit heavily from specific implementations.

Two problem classes stand out in this regard: (i) Extremely large problems; if we want to "boldly go where no man has gone before", frameworks might prove too restrictive to handle the special issues that arise, see Applegate et al. ( ). (ii) Problems where most of the time is spent in specific algorithms for lower and upper bounding and where branching is not an important issue, e. g. Steiner trees in graphs.

In fact, the approach using a modeling language is so fast, it should be used as a fast prototype before engaging in the e ort needed by the other approaches. With the steady progress in solver capabilities (Bixby et al., , Bixby,) many problems can already be solved at this stage. If not, it is a very nice way to make a proof of concept and to investigate di erent modeling approaches on smaller instances.

1.2 What's next?

The next chapter will introduce mathematical modeling languages in general and the Z modeling language in particular. We will continue in Chapter and show how Z is implemented and how we try to make sure the number of bugs in the code will decrease strictly monotone in time. Beginning with Chapter we will describe our successes and problems at some real-world projects. Especially Chapter will show how crucial it is to assess the quality of the data. Models for the problems involved will be discussed for all projects. Finally, in Chapter , we will "revisit" a well-known hard combinatorial problem, namely the Steiner tree packing problem in graphs. We will use a known, but up to now not practically used, formulation to model the classical switchbox-routing problem combined with via-minimization.

If not otherwise stated, all computations were performed on a Dell Precision workstation with a . gigahertz - and gigabytes of . The [4] rating for this kind of computer is listed as SPECint = and SPECfp = .

[4] http://www.spec.org

Part I

Design and Implementation

Chapter 2

The Zimpl Modeling Language

When someone says:
"I want a programming language in which I need
only say what I wish done,"
give him a lollipop.
— fortune( )

2.1 Introduction

Consider the following linear program:

$$\begin{array}{rll} \min & 2x + 3y & \\ \text{subject to} & x + y & \leqslant 6 \\ & x, y & \geqslant 0 \end{array}$$

The standard format used to feed such a problem into a solver is called . in-vented it for the Mathematical Programming System/ (Kallrath, a, Spielberg,) in the sixties. Nearly all available and solvers can read this format. While is a nice format to punch into a punch card and at least a reasonable format to read for a computer, it is quite unreadable for humans. For instance, the file of the above linear program looks as shown in Figure . .

For this reason the development of tools to generate files from human readable problem descriptions started soon after the invention of the format, as described by Kallrath (b), which also contains information on the history of modeling languages and a comprehensive survey on the current state of the art.

Beginning in the eighties algebraic modeling languages like (Bisschop and Meeraus, , Bussieck and Meeraus,) and (Fourer et al., , a) were developed. With these tools it became possible to state an in near mathematical notation and have it automatically translated into format or directly fed into the

```
NAME          ex1.mps
ROWS
 N  OBJECTIV
 L  c1
COLUMNS
    x         OBJECTIV          2
    x         c1                1
    y         OBJECTIV          3
    y         c1                1
RHS
    RHS       c1                6
BOUNDS
 LO BND       x                 0
 LO BND       y                 0
ENDATA
```

Figure . : file format example

appropriate solver. For example, a linear program like

$$\begin{array}{rl} \min & \sum_{i \in I} c_i x_i \\ \text{subject to} & \sum_{i \in I} x_i \leqslant 6 \\ & x_i \geqslant 0 \quad \text{for all } i \in I \end{array}$$

can be written in as

```
set          I;
param        c {I};
var          x {i in I} >= 0;
minimize     cost: sum {i in I} c[i] * x[i];
subject to cons: sum {i in I} x[i] <= 6;
```

So, if could do this in why would one bother to write a new program to do the same in ? The reason lies in the fact that all major modeling languages are commercial products. None of these languages is available as source code for further development. None can be given to colleagues or used in classes, apart from very limited "student editions". Usually only a limited number of operating systems and architectures is supported, sometimes only Microsoft Windows. None will run on any non- architecture system. In Table . we have listed all modeling languages for linear and mixed integer programs described in Kallrath (b).[1]

The situation has improved since when the development of Z started. In at least one other open source modeling system is available, namely the - language, which implements a subset of the language and is part of the linear programming kit.

1 Three entries from the proceedings are omitted: , the Optimization Subroutine Library (also named: Optimization Solutions and Library), because does not include a modeling language and the product is discontinued. The - language for global optimization problems because it is no longer developed or maintained. Finally is omitted, because it is a modeling language for nonlinear programs with automatic di erentiation, which is beyond our scope.

Name		URL	Solver	State
	Advanced Integrated Multi-dimensional Modeling Software	www.aimms.com	open	commercial
	A Modeling Language for Mathematical Programming	www.ampl.com	open	commercial
	General Algebraic Modeling System	www.gams.com	open	commercial
	Lingo	www.lindo.com	fixed	commercial
	(Linear|Logical|Literate) Programming Language	www.virtual-optima.com	open	commercial
	Mixed Integer Non-linear Optimizer	titan.princeton.edu/MINOPT	open	mixed
	Mosel	www.dashoptimization.com	fixed	commercial
	Mathematical Programming Language	www.maximalsoftware.com	open	commercial
	Omni	www.haverly.com	open	commercial
	Optimization Programming Language	www.ilog.com	fixed	commercial
-	GNU Mathematical Programming Language	www.gnu.org/software/glpk	fixed	free
	Zuse Institute Mathematical Programming Language	www.zib.de/koch/zimpl	open	free

Table . : Modeling languages

The current trend in commercial modeling languages is to further integrate features like data base query tools, solvers, report generators, and graphical user interfaces. To date the freely available modeling languages are no match in this regard. Z implements maybe twenty percent of the functionality of . Nevertheless, having perhaps the most important twenty percent proved to be su cient for many real-world projects, as we will see in the second part of this thesis.

According to Cunningham and Schrage ( ), the languages shown in Table . can be separated into two classes, namely the *independent modeling languages*, which do not rely on a specific solver and the *solver modeling languages*, which are deeply integrated with a specific solver. The latter tend to be “real” programming languages which allow the implementation of cut separation and column generation schemes.

Integrating a solver is a major problem for all independent modeling languages because it either requires recompilation or a plug-in architecture. Z does not rely on any specific solver. In fact Z does not integrate with any solver at all, instead it writes an file which any solver can read. In this regard Z could be seen as an advanced matrix generator. On the other hand it is possible to combine Z with the solver by the use of pipes, which eliminates the need to write (possibly very large) files to disk.

Unique features in Zimpl

What makes Z special is the use of rational arithmetic. With a few noted exceptions all computations in Z are done with infinite precision rational arithmetic, ensuring that no rounding errors can occur.

What benefits does this have? Usually, after generating the problem, either the modeling language (Fourer and Gay, ,), or the solver will apply some so-called *preprocessing* or *presolving* to reduce the size of the problem. When implemented with limited precision floating-point arithmetic, it is not too unlikely that problems on the edge of feasibility or with ill-scaled constraints do not give the same results after presolving as solving the unaltered problem. To give a simple contrived example, consider the following linear program:

$$
\begin{array}{rll}
\min & x & \\
\text{subject to} & 10^{-14}x - 10^{-14}y & \geqslant 0 \\
& x + y & \geqslant 4 \\
& x, y & \geqslant 0
\end{array}
$$

[2] . reports $x = 0$ and $y = 4$ as optimal “solution”, even though the is infeasible. The reason is presumably that removes all coe cients whose absolute value is smaller than 10^{-13}. In contrast, the preprocessing in Z performs only mathematically valid transformations.

There are more problems with numerics in optimization. Virtually all solvers use floating-point arithmetic and have various tolerances that influence the algorithm. Nu-

2 http://www.ilog.com/products/cplex

merical inaccuracy lurks everywhere and strictly speaking there is no proof that any of the solutions is correct. Here is another example. Solving

$$\begin{array}{rll} \max & x + 2y & \\ \text{subject to} & x + y & \leqslant 0.0000001 \\ & y & \geqslant 0.0000002 \\ & x & \geqslant 0 \end{array}$$

with results in $x = y = 0$, which is again infeasible. The presolving algorithm completely eliminates the problem and declares it feasible. We get the same result even if presolving is turned o , which is not surprising since the feasibility tolerance within defaults to 10^{-6}. But it becomes clear that nested occurrences of fixable variables might change the feasibility of a problem.

In Koch ( ) the results of on the collection were checked with [3], a program that again uses rational arithmetic to check feasibility and optimality of bases. In several cases the bases computed by the and solvers where either non-optimal or in one case even (slightly) infeasible. We conducted a similar experiment with the mixed integer programming instances from - [4]. Using six solvers, namely [5] version . , - [6] Optimizer release . , the current development version of [7] as of September , [8] version . . , [9] version . and _ [10] version . , bases for the relaxation of the -instances were computed. Default settings were used in all cases, except for and where the presolving was disabled and _ where the following settings were used: –bfp libbfp_LUSOL.so -s -si -se.

The results can be seen in Table . ; listed are all instances, where at least two programs did not succeed in finding a feasible and optimal solution. Optimal means no variable has negative reduced costs. Columns labeled *Feas.* indicate whether the computed basis was feasible. Columns labeled *Opti.* indicate whether the basis was optimal. $\surd$ stands for *yes*, *no* is represented by $\otimes$. A — is shown in case no basis was computed by the program because of algorithmic or numerical failures or exceeding two hours of computing time. In one case a singular basis was produced.

can compute an approximation κ of the condition number of the basis. The column labeled *C* lists $\lfloor \log_{10} \kappa \rfloor$, e. g., $5.8 \cdot 10^6$ is shown as 6. With a few exceptions the condition number estimates of the instances in the table are the highest ones in the whole - . The geometric mean for all instances is . . Even though it is certainly possible to produce optimal feasible solutions for at least some of the problematic instances by using di erent solver settings, it is clear that non-optimal or infeasible solutions are relatively common, especially on numerically di cult instances.

3 http://www.zib.de/koch/perplex
4 http://miplib.zib.de
5 http://www.ilog.com/products/cplex
6 http://www.dashoptimization.com
7 http://www.zib.de/Optimization/Software/Soplex
8 http://www.coin-or.org
9 http://www.gnu.org/software/glpk
10 http://groups.yahoo.com/group/lp_solve

Instance	CPLEX		XPress		SoPlex		Clp		GLPK		lp_solve		C
	Feas.	Opti.	Feas.	Opti.	Feas.	Opti.	Feas.	Opti.	Feas.	Opti.	Feas.	Opti.	
arki001	√	⊗	√	⊗	√	⊗	√	⊗	√	⊗	√	⊗	7
atlanta-ip	√	√	√	⊗	√	⊗	√	⊗	√	⊗	√	⊗	6
dano3mip	√	√	√	⊗	√	⊗	√	√	√	√	√	√	7
harp2	√	√	√	⊗	√	√	√	√	√	√	√	⊗	5
momentum1	√	√	⊗	⊗	√	√	√	√	√	√	√	⊗	6
momentum2	⊗	⊗	⊗	⊗	√	⊗	⊗	√	√	√	√	⊗	7
momentum3	⊗	⊗	⊗	⊗	⊗	⊗	⊗	√	⊗	√	—		7
msc98-ip	√	⊗	⊗	⊗	√	⊗	√	⊗	√	⊗	—		4
mzzv11	√	√	⊗	⊗	√	√	√	√	√	√	—		5
qiu	⊗	√	⊗	√	⊗	√	⊗	√	√	√	√	√	4
roll3000	√	√	—		√	√	√	√	√	√	√	√	6
stp3d	√	√	√	√	√	√	√	√	—		—		6

Table . : Results of solving the root relaxation of instances from -

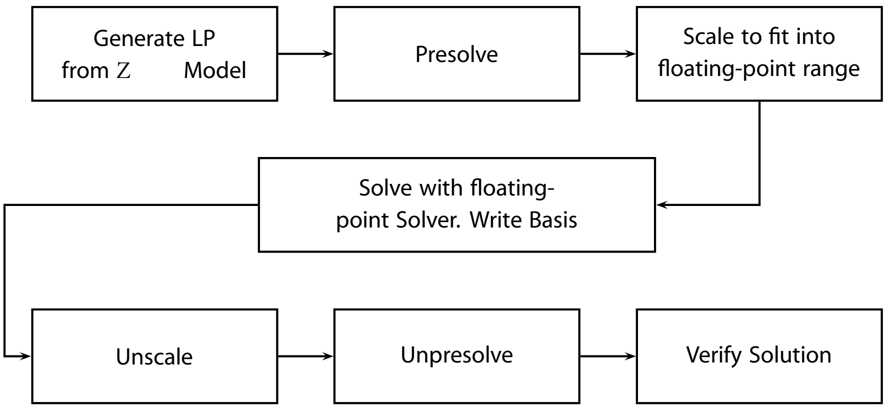


Figure . : Computing exact solutions

Now in combination with Z it will be possible to model a problem, apply presolving and scale the constraint matrix to maximize the usable precision of the floating-point arithmetic, all using rational arithmetic. After writing the resulting linear program both as a restricted precision -file, and in a format that allows unlimited precision, the -file can be used by a standard solver to compute a (hopefully) optimal basis. This basis together with the unlimited precision description of the can be used to unscale and undo the presolving again using rational arithmetic. Finally - can verify the feasibility and optimality of the solution and compute the precise values of the decision variables. Figure . gives an overview.

There is no guarantee that the floating-point solver is able to find a feasible or optimal basis at all. Currently we only have heuristic remedies in this case. One is to change the thresholds within the solver, e. g., decreasing the feasibility tolerance. Another is to use a solver that employs -bit instead of the usual -bit floating-point arithmetic. Preliminary computational experiments with a special -bit version of (Wunderling,) are promising in this regard. A more general solution is to extend to use the basis supplied by the floating-point solver as a starting point and proceed with an all-rational-arithmetic simplex algorithm.

In the next three sections, we will show how to run Z , describe the language in detail and give short examples on how to model some classic combinatorial problems like the traveling salesman or the n-queens problem.

2.2 Invocation

In order to run Z on a model given in the file `ex1.zpl` type the command:

```
zimpl ex1.zpl
```

In general terms the command is:

```
zimpl [options] <input-files>
```

It is possible to give more than one input file. They are read one after the other as if they were all one big file. If any error occurs while processing, Z prints out an error message and aborts. In case everything goes well, the results are written into two or more files, depending on the specified options.

-t *format*	Selects the output format. Can be either `lp`, which is default, or `mps`, or `hum`, which is only human readable.
-o *name*	Sets the base-name for the output files. Defaults to the name of the first input file with its path and extension stripped off.
-F *filter*	The output is piped through a filter. A %s in the string is replaced by the output filename. For example `-F "gzip -c >%s.gz"` would compress all the output files.
-n *cform*	Select the format for the generation of constraint names. Can be `cm`, which will number them $1 \dots n$ with a 'c' in front. `cn` will use the name supplied in the `subto` statement and number them $1 \dots n$ within the statement. `cf` will use the name given with the `subto`, then a $1 \dots n$ number like in `cm` and then append all the local variables from the forall statements.
-v *1..5*	Set the verbosity level. 0 is quiet, 1 is default, 2 is verbose, 3 and 4 are chatter, and 5 is debug.
-D *name=val*	Sets the parameter *name* to the specified value. This is equivalent with having this line in the Z program: `param name:=val`.
-b	Enables bison debug output.
-f	Enables flex debug output.
-h	Prints a help message.
-m	Writes a CPLEX `mst` (Mip STart) file.
-O	Try to reduce the generated LP by doing some presolve analysis.
-r	Writes a CPLEX `ord` branching order file.
-V	Prints the version number.

Table . : Z options

The first output file is the problem generated from the model in either , , or a "human readable" format, with extensions *.lp*, *.mps*, or *.hum*, respectively. The next one is the *table* file, which has the extension *.tbl*. The table file lists all variable and constraint names used in the model and their corresponding names in the problem file. The reason for this name translation is the limitation of the length of names in the format to eight characters. Also the format restricts the length of names. The precise limit is depending on the version. . has a limit of characters, and ignores silently the rest of the name, while . has a limit of characters, but will for some commands only show the first characters in the output.

A complete list of all options understood by Z can be found in Table . . A typical invocation of Z is for example:

```
zimpl -o solveme -t mps data.zpl model.zpl
```

This reads the files data.zpl and model.zpl as input and produces as output the files solveme.mps and solveme.tbl. Note that in case -output is specified for a maximization problem, the objective function will be inverted, because the format has no provision for stating the sense of the objective function. The default is to assume maximization.

2.3 Format

Each -file consists of six types of statements:

- Sets
- Parameters
- Variables
- Objective
- Constraints
- Function definitions

Each statement ends with a semicolon. Everything from a hash-sign `#`, provided it is not part of a string, to the end of the line is treated as a comment and is ignored. If a line starts with the word `include` followed by a filename in double quotation marks, then this file is read and processed instead of the line.

2.3.1 Expressions

Z works on its lowest level with two types of data: Strings and numbers. Wherever a number or string is required it is also possible to use a parameter of the corresponding value type. In most cases expressions are allowed instead of just a number or a string. The precedence of operators is the usual one, but parentheses can always be used to specify the evaluation order explicitly.

Numeric expressions

A number in Z can be given in the usual format, e. g. as , - . or . e- . Numeric expressions consist of numbers, numeric valued parameters, and any of the operators and functions listed in Table . . Additionally the functions shown in Table . can be used. Note that those functions are only computed with normal double precision floating-point arithmetic and therefore have limited accuracy.

`a^b`, `a**b`	a to the power of b	a^b
`a+b`	addition	$a + b$
`a-b`	subtraction	$a - b$
`a*b`	multiplication	$a \cdot b$
`a/b`	division	a/b
`a mod b`	modulo	$a \mod b$
`abs(a)`	absolute value	$\|a\|$
`sgn(a)`	sign	$x > 0 \Rightarrow 1, x < 0 \Rightarrow -1$, else 0
`floor(a)`	round down	$\lfloor a \rfloor$
`ceil(a)`	round up	$\lceil a \rceil$
`a!`	factorial	$a!$
`min(S)`	minimum of a set	$\min_{s \in S}$
`max(S)`	maximum of a set	$\max_{s \in S}$
`min(a,b,c,...,n)`	minimum of a list	$\min(a, b, c, \ldots, n)$
`max(a,b,c,...,n)`	maximum of a list	$\max(a, b, c, \ldots, n)$
`card(S)`	cardinality of a set	$\|S\|$
`ord(A,n,c)`	ordinal	c-th component of the n-th element of set A.
`if a then b else c end`	conditional	$\begin{cases} b, & \text{if } a = \text{true} \\ c, & \text{if } a = \text{false} \end{cases}$

Table . : Rational arithmetic functions

`sqrt(a)`	square root	$\sqrt{a}$
`log(a)`	logarithm to base 10	$\log_{10} a$
`ln(a)`	natural logarithm	$\ln a$
`exp(a)`	exponential function	e^a

Table . : Double precision functions

String expressions

A string is delimited by double quotation marks `"`, e. g. `"Hallo Keiken"`.

Variant expressions

The following is either a numeric or a string expression, depending on whether *expression* is a string or a numeric expression:

`if` *boolean-expression* `then` *expression* `else` *expression* `end`

The same is true for the `ord(`*set, tuple-number, component-number*`)` function, which evaluates to a specific element of a set (details about sets are covered below).

Boolean expressions

These evaluate either to *true* or to *false*. For numbers and strings the relational operators <, <=, ==, !=, >=, and > are defined. Combinations of Boolean expressions with `and`, `or`, and `xor` [11] and negation with `not` are possible. The expression *tuple* `in` *set-expression* (explained in the next section) can be used to test set membership of a tuple.

2.3.2 Sets

Sets consist of tuples. Each tuple can only be once in a set. The sets in Z are all ordered, but there is no particular order of the tuples. Sets are delimited by braces, `{` and `}`, respectively. Tuples consist of components. The components are either numbers or strings. The components are ordered. All tuples of a specific set have the same number of components. The type of the n-th component for all tuples of a set must be the same, i. e., they have to be either all numbers or all strings. The definition of a tuple is enclosed in angle brackets < and >, e. g. `<1,2,"x">`. The components are separated by commas. If tuples are one-dimensional, it is possible to omit the tuple delimiters in a list of elements, but in this case they must be omitted from all tuples in the definition, e. g. `{1,2,3}` is valid while `{1,2,<3>}` is not.

Sets can be defined with the set statement. It consists of the keyword `set`, the name of the set, an assignment operator `:=` and a valid set expression.

Sets are referenced by the use of a *template* tuple, consisting of placeholders, which are replaced by the values of the components of the respective tuple. For example, a set S consisting of two-dimensional tuples could be referenced by `<a,b> in S`. If any of the placeholders are actual values, only those tuples matching these values will be extracted. For example, `<1,b> in S` will only get those tuples whose first component is `1`. Please note that if one of the placeholders is the name of an already defined parameter, set or variable, it will be substituted. This will result either in an error or an actual value.

Examples

```
set A := { 1, 2, 3 };
set B := { "hi", "ha", "ho" };
set C := { <1,2,"x">, <6,5,"y">, <787,12.6,"oh"> };
```

For set expressions the functions and operators given in Table . are defined.

An example for the use of the `if` *boolean-expression* `then` *set-expression* `else` *set-expression* `end` can be found on page together with the examples for indexed sets.

11 $a \text{ xor } b := a \wedge \neg b \vee \neg a \wedge b$

Examples

```
set D := A cross B;
set E := { 6 to 9 } union A without { 2, 3 };
set F := { 1 to 9 } * { 10 to 19 } * { "A", "B" };
set G := proj(F, <3,1>);
# will give: { <"A",1>, <"A",2"> ... <"B",9> }
```

`A*B`, `A cross B`	cross product	$\{(x,y) \mid x \in A \wedge y \in B\}$
`A+B`, `A union B`	union	$\{x \mid x \in A \vee x \in B\}$
`A inter B`	intersection	$\{x \mid x \in A \wedge x \in B\}$
`A\B`, `A-B`, `A without B`	difference	$\{x \mid x \in A \wedge x \notin B\}$
`A symdiff B`	symmetric difference	$\{x \mid (x \in A \wedge x \notin B) \vee (x \in B \wedge x \notin A)\}$
`{n .. m}`, `{n to m` *by s*`}`	generate, (default $s = 1$)	$\{x \mid x = n + is \leqslant m, i \in \mathbb{N}_0, x, n, m, s \in \mathbb{Z}\}$
`proj(A, t)`	projection $t = (e_1, \ldots, e_n)$	The new set will consist of n-tuples, with the i-th component being the e_i-th component of A.
`if a then b else c end`	conditional	$\begin{cases} b, & \text{if } a = \text{true} \\ c, & \text{if } a = \text{false} \end{cases}$

Table . : Set related functions

Conditional sets

It is possible to restrict a set to tuples that satisfy a Boolean expression. The expression given by the `with` clause is evaluated for each tuple in the set and only tuples for which the expression evaluates to *true* are included in the new set.

Examples

```
set F := { <i,j> in Q with i > j and i < 5 };
set A := { "a", "b", "c" };
set B := { 1, 2, 3 };
set V := { <a,2> in A*B with a == "a" or a == "b" };
# will give: { <"a",2>, <"b",2> }
```

Indexed sets

It is possible to index one set with another set resulting in a set of sets. Indexed sets are accessed by adding the index of the set in brackets `[` and `]`, like `S[7]`. Table . lists the available functions. There are three possibilities how to assign to an indexed set:

- The assignment expression is a list of comma-separated pairs, consisting of a tuple from the index set and a set expression to assign.
- If an index tuple is given as part of the index, e. g. <i> in I, the assignment is evaluated for each value of index tuple.
- By use of a function that returns an indexed set.

Examples

```
set I           := { 1..3 };
set A[I]        := <1> {"a","b"}, <2> {"c","e"}, <3> {"f"};
set B[<i> in I] := { 3 * i };
set P[]         := powerset(I);
set J           := indexset(P);
set S[]         := subset(I, 2);
set K[<i> in I] := if i mod 2 == 0 then { i } else { -i } end;
```

powerset(A)	generates all subsets of A	$\{X \mid X \subseteq A\}$
subset(A,n)	generates all subsets of A with n elements	$\{X \mid X \subseteq A \land \lvert X\rvert = n\}$
indexset(A)	the index set of A	$\{1 \ldots \lvert A\rvert\}$

Table . : Indexed set functions

2.3.3 Parameters

Parameters can be declared with or without an index set. Without indexing a parameter is just a single value, which is either a number or a string. For indexed parameters there is one value for each member of the set. It is possible to declare a *default* value.

Parameters are declared in the following way: The keyword param is followed by the name of the parameter optionally followed by the index set. Then after the assignment sign comes a list of pairs. The first element of each pair is a tuple from the index set, while the second element is the value of the parameter for this index.

Examples

```
set A := { 12 .. 30 };
set C := { <1,2,"x">, <6,5,"y">, <3,7,"z" };
param q := 5;
param u[A] := <13> 17, <17> 29, <23> 12 default 99;
param w[C] := <1,2,"x"> 1/2, <6,5,"y"> 2/3;
param x[<i> in { 1 .. 8 } with i mod 2 == 0] := 3 * i;
```

Assignments need not to be complete. In the example, no value is given for index <3,7,"z"> of parameter w. This is correct as long as it is never referenced.

Parameter tables

It is possible to initialize multi-dimensional indexed parameters from tables. This is especially useful for two-dimensional parameters. The data is put in a table structure with | signs on each margin. Then a headline with column indices has to be added, and one index for each row of the table is needed. The column index has to be one-dimensional, but the row index can be multi-dimensional. The complete index for the entry is built by appending the column index to the row index. The entries are separated by commas. Any valid expression is allowed here. As can be seen in the third example below, it is possible to add a list of entries after the table.

Examples

```
set I := { 1 .. 10 };
set J := { "a", "b", "c", "x", "y", "z" };

param h[I*J] :=   | "a", "c", "x", "z"    |
                |1|  12,  17, 99,     23 |
                |3|   4,   3,-17, 66*5.5 |
                |5| 2/3, -.4,  3, abs(-4)|
                |9|   1,   2,  0,      3 | default -99;

param g[I*I*I] :=      | 1, 2, 3 |
                  |1,3| 0, 0, 1 |
                  |2,1| 1, 0, 1 |;

param k[I*I] :=  |  7,  8,  9 |
               |4| 89, 67, 55 |
               |5| 12, 13, 14 |, <1,2> 17, <3,4> 99;
```

The last example is equivalent to:

```
param k[I*I] := <4,7> 89, <4,8> 67, <4,9> 44, <5,7> 12,
                <5,8> 13, <5,9> 14, <1,2> 17, <3,4> 99;
```

2.3.4 Variables

Like parameters, variables can be indexed. A variable has to be one out of three possible types: Continuous (called *real*), binary or integer. The default type is real. Variables may have lower and upper bounds. Defaults are zero as lower and infinity as upper bound. Binary variables are always bounded between zero and one. It is possible to compute the value of the lower or upper bounds depending on the index of the variable (see the last declaration in the example). Bounds can also be set to `infinity` and `-infinity`.

Examples

```
var x1;
var x2 binary;
var y[A] real >= 2 <= 18;
var z[<a,b> in C] integer
      >= a * 10 <= if b <= 3 then p[b] else 10 end;
```

2.3.5 Objective

There must be at most one objective statement in a model. The objective can be either `minimize` or `maximize`. Following the keyword is a name, a colon : and then a linear term expressing the objective function.

Example

```
minimize cost: 12 * x1 -4.4 * x2
   + sum <a> in A : u[a] * y[a]
   + sum <a,b,c> in C with a in E and b > 3 : -a/2 * z[a,b,c];
maximize profit: sum <i> in I : c[i] * x[i];
```

2.3.6 Constraints

The general format for a constraint is:

```
subto name: term sense term
```

Alternatively it is also possible to define *ranged* constraints, which have the form:

```
name: expr sense term sense expr
```

`name` can be any name starting with a letter. `term` is defined as in the objective. `sense` is one of <=, >= and ==. In case of ranged constraints both senses have to be equal and may not be ==. `expr` is any valid expression that evaluates to a number. Many constraints can be generated with one statement by the use of the `forall` instruction, as shown below.

Examples

```
subto time:  3 * x1 + 4 * x2 <= 7;
subto space: 50 >= sum <a> in A: 2 * u[a] * y[a] >= 5;
subto weird: forall <a> in A: sum <a,b,c> in C: z[a,b,c] == 55;
subto c21: 6 * (sum <i> in A: x[i] + sum <j> in B : y[j]) >= 2;
subto c40: x[1] == a[1] + 2 * sum <i> in A do 2*a[i]*x[i]*3 + 4;
```

2.3.7 Details on *sum* and *forall*

The general forms are:

`forall` *index* `do` *term* and `sum` *index* `do` *term*

It is possible to nest several forall instructions. The general form of *index* is:

tuple `in` *set* `with` *boolean-expression*

It is allowed to write a colon `:` instead of `do` and a vertical bar `|` instead of `with`. The number of components in the *tuple* and in the members of the *set* must match. The `with` part of an *index* is optional. The *set* can be any expression giving a set.

Examples

```
forall <i,j> in X cross { 1 to 5 } without { <2,3> }
   with i > 5 and j < 2 do
      sum <i,j,k> in X cross { 1 to 3 } cross Z do
         p[i] * q[j] * w[j,k] >= if i == 2 then 17 else 53;
```

Note that in the example *i* and *j* are set by the `forall` instruction. So they are fixed in all invocations of `sum`.

2.3.8 Details on *if* in constraints

It is possible to put two variants of a constraint into an `if`-statement. The same applies for *terms*. A `forall` statement inside the result part of an `if` is also possible.

Examples

```
subto c1: forall <i> in I do
  if (i mod 2 == 0) then  3 * x[i] >= 4
                    else -2 * y[i] <= 3 end;

subto c2: sum <i> in I :
     if (i mod 2 == 0) then 3 * x[i] else -2 * y[i] end <= 3;
```

2.3.9 Initializing sets and parameters from a file

It is possible to load the values for a set or a parameter from a file. The syntax is:

`read` *filename* `as` *template* [`skip` *n*] [`use` *n*] [`fs` *s*] [`comment` *s*]

filename is the name of the file to read. *template* is a string with a template for the tuples to generate. Each input line from the file is split into fields. The splitting is done according to the following rules: Whenever a space, tab, comma, semicolon or double colon is encountered a new field is started. Text that is enclosed in double quotes is not split and the quotes are always removed. When a field is split all space and tab characters around the splitting point are removed. If the split is due to a comma, semicolon or double colon, each occurrence of these characters starts a new field.

Examples

All these lines have three fields:

```
Hallo;12;3
Moin   7  2
"Hallo, Peter"; "Nice to meet you" 77
,,2
```

For each component of the tuple, the number of the field to use for the value is given, followed by either `n` if the field should be interpreted as a number or `s` for a string. After the template some optional modifiers can be given. The order does not matter. `skip` *n* instructs to skip the first *n* lines of the file. `use` *n* limits the number of lines to use to *n*. `comment` *s* sets a list of characters that start comments in the file. Each line is ended when any of the comment characters is found. When a file is read, empty lines are skipped and not counted for the `use` clause. They are counted for the `skip` clause.

Examples

```
set P := { read "nodes.txt" as "<1s>" };
```

```
nodes.txt:
   Hamburg              → <"Hamburg">
   München              → <"München">
   Berlin               → <"Berlin">
```

```
set Q := { read "blabla.txt" as "<1s,5n,2n>" skip 1 use 2 };
```

```
blabla.txt:
   Name;Nr;X;Y;No       → skip
   Hamburg;12;x;y;7     → <"Hamburg", ,  >
   Bremen;4;x;y;5       → <"Bremen, , >
   Berlin;2;x;y;8       → skip
```

```
param cost[P] := read "cost.txt" as "<1s> 2n" comment "#";
```

```
cost.txt:
   # Name Price         → skip
   Hamburg 1000         → <"Hamburg">
   München 1200         → <"München">
   Berlin  1400         → <"Berlin">
```

```
param cost[Q] := read "haha.txt" as "<3s,1n,2n> 4s";
```

```
haha.txt:
      1:2:ab:con1       → <"ab", , > "con "
      2:3:bc:con2       → <"bc", , > "con "
      4:5:de:con3       → <"de", , > "con "
```

As with table format input, it is possible to add a list of tuples or parameter entries after a read statement.

Examples

```
set A := { read "test.txt" as "<2n>", <5>, <6> };
param winniepoh[X] :=
   read "values.txt" as "<1n,2n> 3n", <1,2> 17, <3,4> 29;
```

2.3.10 Function definitions

It is possible to define functions within Z . The value a function returns has to be either a number, a string or a set. The arguments of a function can only be numbers or strings, but within the function definition it is possible to access all otherwise declared sets, parameters and variables.

The definition of a function has to start with `defnumb`, `defstrg` or `defset`, depending on the return value. Then follows the name of the function and a list of argument names put in parentheses. Next is an assignment operator := and a valid expression or set expression.

Examples

```
defnumb dist(a,b)   := sqrt(a*a + b*b);
defstrg huehott(a) := if a < 0 then "hue" else "hott" end;
defset  bigger(i)   := { <j> in K with j > i };
```

2.3.11 Extended constraints

Z has the possibility to generate systems of constraints that mimic conditional constraints. The general syntax is as follows (note that the `else` part is optional):

`vif` *boolean-constraint* `then` *constraint* [`else` *constraint*] `end`

where *boolean-constraint* consists of a linear expression involving variables. All these variables have to be bounded integer or binary variables. It is not possible to use any continuous variables or integer variables with infinite bounds in a *boolean-constraint*. All comparison operators ($<$, $\leqslant$, $==$, $!=$, $\geqslant$, $>$) are allowed. Also combination of several terms with `and`, `or`, and `xor` and negation with `not` is possible. The conditional constraints (those which follow after `then` or `else`) may include bounded continuous variables. Be aware that using this construct will lead to the generation of several additional constraints and variables.

Examples

```
var x[I] integer >= 0 <= 20;
subto c1: vif 3 * x[1] + x[2] != 7
   then sum <i> in I : y[i] <= 17
```

```
   else sum <k> in K : z[k] >= 5 end;
subto c2: vif x[1] == 1 and x[2] > 5 then x[3] == 7 end;
subto c3: forall <i> in I with i < max(I) :
   vif x[i] >= 2 then x[i + 1] <= 4 end;
```

2.3.12 Extended functions

It is possible to use special functions on terms with variables that will automatically be converted into a system of inequalities. The arguments of these functions have to be linear terms consisting of bounded integer or binary variables. At the moment only the function `vabs(t)` that computes the absolute value of the term `t` is implemented, but functions like the minimum or the maximum of two terms, or the sign of a term can be implemented in a similar manner. Again, using this construct will lead to the generation of several additional constraints and variables.

Examples

```
var x[I] integer >= -5 <= 5;
subto c1: vabs(sum <i> in I : x[i]) <= 15;
subto c2: vif vabs(x[1] + x[2]) > 2 then x[3] == 2 end;
```

2.3.13 The *do print* and *do check* commands

The `do` command is special. It has two possible incarnations: `print` and `check`. `print` will print to the standard output stream whatever numerical, string, Boolean or set expression, or tuple follows it. This can be used for example to check if a set has the expected members, or if some computation has the anticipated result. `check` always precedes a Boolean expression. If this expression does not evaluate to *true*, the program is aborted with an appropriate error message. This can be used to assert that specific conditions are met. It is possible to use a `forall` clause before a `print` or `check` statement.

Examples

```
set I := { 1..10 };
do print I;
do forall <i> in I with i > 5 do print sqrt(i);
do forall <p> in P do check sum <p,i> in PI : 1 >= 1;
```

2.4 Modeling examples

In this section we show some examples of well-known problems translated into Z format.

2.4.1 The diet problem

This is the first example in Chvátal (, Chapter , page). It is a classic so-called *diet* problem, see for example Dantzig ( ) about its implications in practice.

Given a set of foods F and a set of nutrients N, we have a table π_{fn} of the amount of nutrient n in food f. Now Π_n defines how much intake of each nutrient is needed. Δ_f denotes for each food the maximum number of servings acceptable. Given prices c_f for each food, we have to find a selection of foods that obeys the restrictions and has minimal cost. An integer variable x_f is introduced for each $f \in F$ indicating the number of servings of food f. Integer variables are used, because only complete servings can be obtained, i. e., half an egg is not an option. The problem may be stated as:

$$
\begin{aligned}
\min \sum_{f \in F} c_f x_f & \qquad \text{subject to} \\
\sum_{f \in F} \pi_{fn} x_f \geqslant \Pi_n & \qquad \text{for all } n \in N \\
0 \leqslant x_f \leqslant \Delta_f & \qquad \text{for all } f \in F \\
x_f \in \mathbb{N}_0 & \qquad \text{for all } f \in F \qquad (\ . \ )
\end{aligned}
$$

This translates into Z as follows:

```
set Food      := { "Oatmeal", "Chicken", "Eggs",
                   "Milk",    "Pie",     "Pork" };
set Nutrients := { "Energy",  "Protein", "Calcium" };
set Attr      := Nutrients + { "Servings", "Price" };

param needed[Nutrients] :=
   <"Energy"> 2000, <"Protein"> 55, <"Calcium"> 800;

param data[Food * Attr] :=
           |"Servings","Energy","Protein","Calcium","Price"|
|"Oatmeal"|         4 ,    110 ,       4 ,        2 ,     3 |
|"Chicken"|         3 ,    205 ,      32 ,       12 ,    24 |
|"Eggs"   |         2 ,    160 ,      13 ,       54 ,    13 |
|"Milk"   |         8 ,    160 ,       8 ,      284 ,     9 |
|"Pie"    |         2 ,    420 ,       4 ,       22 ,    20 |
|"Pork"   |         2 ,    260 ,      14 ,       80 ,    19 |;
#                         (kcal)      (g)      (mg) (cents)

var x[<f> in Food] integer >= 0 <= data[f, "Servings"];

minimize cost: sum <f> in Food : data[f, "Price"] * x[f];

subto need: forall <n> in Nutrients do
   sum <f> in Food : data[f, n] * x[f] >= needed[n];
```

The cheapest meal satisfying all requirements costs cents and consists of four servings of oatmeal, five servings of milk and two servings of pie.

2.4.2 The traveling salesman problem

In this example we show how to generate an exponential description of the *symmetric traveling salesman problem* ( ) as given for example in Schrijver (, Section .).

Let $G = (V, E)$ be a complete graph, with V being the set of cities and E being the set of links between the cities. Introducing binary variables x_{ij} for each $(i, j) \in E$ indicating if edge (i, j) is part of the tour, the can be written as:

$$
\begin{aligned}
\min \sum_{(i,j)\in E} d_{ij}x_{ij} & & \text{subject to} \\
\sum_{(i,j)\in\delta_v} x_{ij} &= 2 & \text{for all } v \in V \\
\sum_{(i,j)\in E(U)} x_{ij} &\leqslant |U| - 1 & \text{for all } U \subseteq V, \emptyset \neq U \neq V \qquad (\ .\ ) \\
x_{ij} &\in \{0, 1\} & \text{for all } (i, j) \in E
\end{aligned}
$$

The data is read in from a file that gives the number of the city and the x and y coordinates. Distances between cities are assumed Euclidean. For example:

```
# City        X    Y
Berlin      5251 1340
Frankfurt   5011  864
Leipzig     5133 1237
Heidelberg  4941  867
Karlsruhe   4901  840
Hamburg     5356  998
Bayreuth    4993 1159
Trier       4974  668
Hannover    5237  972
Stuttgart   4874  909
Passau      4856 1344
Augsburg    4833 1089
Koblenz     5033  759
Dortmund    5148  741
Bochum      5145  728
Duisburg    5142  679
Wuppertal   5124  715
Essen       5145  701
Jena        5093 1158
```

The formulation in Z follows below. Please note that `P[]` holds all subsets of the cities. As a result cities is about as far as one can get with this approach. Information on how to solve much larger instances can be found on the website[12].

```
set V               := { read "tsp.dat" as "<1s>" comment "#" };
set E               := { <i,j> in V * V with i < j };
set P[]             := powerset(V);
set K               := indexset(P);
param px[V]         := read "tsp.dat" as "<1s> 2n" comment "#";
param py[V]         := read "tsp.dat" as "<1s> 3n" comment "#";
defnumb dist(a,b) := sqrt((px[a]-px[b])^2 + (py[a]-py[b])^2);

var x[E] binary;

minimize cost: sum <i,j> in E : dist(i,j) * x[i, j];

subto two_connected: forall <v> in V do
   (sum <v,j> in E : x[v,j]) + (sum <i,v> in E : x[i,v]) == 2;
```

12 http://www.tsp.gatech.edu

```
subto no_subtour:
   forall <k> in K with
      card(P[k]) > 2 and card(P[k]) < card(V) - 2 do
      sum <i,j> in E with <i> in P[k] and <j> in P[k] : x[i,j]
      <= card(P[k]) - 1;
```

The resulting LP has variables, , constraints, and , , non-zero entries in the constraint matrix, giving an -file size of . solves this to optimality without branching in less than a minute.[13]

An optimal tour for the data above is Berlin, Hamburg, Hannover, Dortmund, Bochum, Wuppertal, Essen, Duisburg, Trier, Koblenz, Frankfurt, Heidelberg, Karlsruhe, Stuttgart, Augsburg, Passau, Bayreuth, Jena, Leipzig, Berlin.

2.4.3 The capacitated facility location problem

Here we give a formulation of the *capacitated facility location* problem. It may also be considered as a kind of *bin packing* problem with packing costs and variable sized bins, or as a *cutting stock* problem with cutting costs.

Given a set of possible plants P to build, and a set of stores S with a certain demand δ_s that has to be satisfied, we have to decide which plant should serve which store. We have costs c_p for building plant p and c_{ps} for transporting the goods from plant p to store s. Each plant has only a limited capacity κ_p. We insist that each store is served by exactly one plant. Of course we are looking for the cheapest solution:

$$\min \sum_{p\in P} c_p z_p + \sum_{p\in P, s\in S} c_{ps} z_{ps} \qquad \text{subject to}$$

$$\sum_{p\in P} x_{ps} = 1 \qquad \text{for all } s \in S \qquad (\ .\ )$$

$$x_{ps} \leqslant z_p \qquad \text{for all } s \in S, p \in P \qquad (\ .\ )$$

$$\sum_{s\in S} \delta_s x_{ps} \leqslant \kappa_p \qquad \text{for all } p \in P \qquad (\ .\ )$$

$$x_{ps}, z_p \in \{0, 1\} \qquad \text{for all } p \in P, s \in S$$

We use binary variables z_p, which are set to one, if and only if plant p is to be built. Additionally we have binary variables x_{ps}, which are set to one if and only if plant p serves shop s. Equation (.) demands that each store is assigned to exactly one plant. Inequality (.) makes sure that a plant that serves a shop is built. Inequality (.) assures that the shops are served by a plant which does not exceed its capacity. Putting this into Z yields the program shown on the next page. The optimal solution for the instance described by the program is to build plants A and C. Stores , , and are served by plant A and the others by plant C. The total cost is .

13 Only simplex iterations are needed to reach the optimal solution.

```
set PLANTS := { "A", "B", "C", "D" };
set STORES := { 1 .. 9 };
set PS     := PLANTS * STORES;

# How much does it cost to build a plant and what capacity
# will it then have?
param building[PLANTS]:= <"A"> 500, <"B"> 600, <"C"> 700, <"D"> 800;
param capacity[PLANTS]:= <"A">  40, <"B">  55, <"C">  73, <"D">  90;

# The demand of each store
param demand  [STORES]:= <1> 10, <2> 14,
                          <3> 17, <4>  8,
                          <5>  9, <6> 12,
                          <7> 11, <8> 15,
                          <9> 16;

# Transportation cost from each plant to each store
param transport[PS] :=
       |  1,  2,  3,  4,  5,  6,  7,  8,  9 |
   |"A"| 55,  4, 17, 33, 47, 98, 19, 10,  6 |
   |"B"| 42, 12,  4, 23, 16, 78, 47,  9, 82 |
   |"C"| 17, 34, 65, 25,  7, 67, 45, 13, 54 |
   |"D"| 60,  8, 79, 24, 28, 19, 62, 18, 45 |;

var x[PS]     binary;  # Is plant p supplying store s ?
var z[PLANTS] binary;  # Is plant p built ?

# We want it cheap
minimize cost: sum <p> in PLANTS : building[p] * z[p]
             + sum <p,s> in PS : transport[p,s] * x[p,s];

# Each store is supplied by exactly one plant
subto assign:
   forall <s> in STORES do
      sum <p> in PLANTS : x[p,s] == 1;

# To be able to supply a store, a plant must be built
subto build:
    forall <p,s> in PS do
       x[p,s] <= z[p];

# The plant must be able to meet the demands from all stores
# that are assigned to it
subto limit:
    forall <p> in PLANTS do
       sum <s> in S : demand[s] * x[p,s] <= capacity[p];
```

2.4.4 The n-queens problem

The problem is to place n queens on a $n \times n$ chessboard so that no two queens are on the same row, column or diagonal. The n-queens problem is a classic combinatorial search problem often used to test the performance of algorithms that solve satisfiability problems. Note though, that there are algorithms available which need linear time in practise, like, for example, those of Sosič and Gu ( ). We will show four di erent models for the problem and compare their performance.

The integer model

The first formulation uses one general integer variable for each row of the board. Each variable can assume the value of a column, i. e., we have n variables with bounds $1 \dots n$. Next we use the vabs extended function to model an *all different* constraint on the variables (see constraint c). This makes sure that no queen is located on the same column than any other queen. The second constraint (c) is used to block all the diagonals of a queen by demanding that the absolute value of the row distance and the column distance of each pair of queens are di erent. We model $a \neq b$ by $\text{abs}(a - b) \geqslant 1$.

Note that this formulation only works if a queen can be placed in each row, i. e., if the size of the board is at least 4×4.

```
param queens := 8;

set C := { 1 .. queens };
set P := { <i,j> in C * C with i < j };

var x[C] integer >= 1 <= queens;

subto c1: forall <i,j> in P do vabs(x[i] - x[j]) >= 1;
subto c2: forall <i,j> in P do
         vabs(vabs(x[i] - x[j]) - abs(i - j)) >= 1;
```

The following table shows the performance of the model. Since the problem is modeled as a pure satisfiability problem, the solution time depends only on how long it takes to find a feasible solution.[14] The columns titled *Vars*, *Cons*, and *NZ* denote the number of variables, constraints and non-zero entries in the constraint matrix of the generated integer program. *Nodes* lists the number of branch-and-bound nodes evaluated by the solver, and *time* gives the solution time in seconds.

Queens	Vars	Cons	NZ	Nodes	Time [s]
8	344	392	951	1,324	<1
12	804	924	2,243	122,394	120
16	1,456	1,680	4,079	>1 mill.	>1,700

14 Which is, in fact, rather random.

As we can see, between and queens is the maximum instance size we can expect to solve with this model. Neither changing the parameters to aggressive cut generation nor setting emphasis on integer feasibility improves the performance significantly.

The binary models

Another approach to model the problem is to have one binary variable for each square of the board. The variable is one if and only if a queen is on this square and we maximize the number of queens on the board.

For each square we compute in advance which other squares are blocked if a queen is placed on this particular square. Then the extended `vif` constraint is used to set the variables of the blocked squares to zero if a queen is placed.

```
param columns := 8;

set C   := { 1 .. columns };
set CxC := C * C;

set TABU[<i,j> in CxC] := { <m,n> in CxC with (m != i or n != j)
   and (m == i or n == j or abs(m - i) == abs(n - j)) };

var x[CxC] binary;

maximize queens: sum <i,j> in CxC : x[i,j];

subto c1: forall <i,j> in CxC do vif x[i,j] == 1 then
                 sum <m,n> in TABU[i,j] : x[m,n] <= 0 end;
```

Using extended formulations can make the models more comprehensible. For example, replacing constraint c in line with an equivalent one that does not use `vif` as shown below, leads to a formulation that is much harder to understand.

```
subto c2: forall <i,j> in CxC do
             card(TABU[i,j]) * x[i,j]
           + sum <m,n> in TABU[i,j] : x[m,n] <= card(TABU[i,j]);
```

After the application of the presolve procedure both formulations result in identical integer programs. The performance of the model is shown in the following table. *S* indicates the settings used: Either *(D)efault*, *(C)uts*[15], or *(F)easibility*[16]. *Root Node* indicates the objective function value of the relaxation of the root node.

15 Cuts: `mip cuts all 2` and `mip strategy probing 3`.
16 Feasibility: `mip cuts all -1` and `mip emph 1`

Queens	S	Vars	Cons	NZ	Root Node	Nodes	Time [s]
8	D	384	448	2,352	13.4301	241	<1
	C				8.0000	0	<1
12	D	864	1,008	7,208	23.4463	20,911	4
	C				12.0000	0	<1
16	D	1,536	1,792	16,224	35.1807	281,030	1,662
	C				16.0000	54	8
24	C	3,456	4,032	51,856	24.0000	38	42
32	C	6,144	7,168	119,488	56.4756	$>5{,}500$	$>2{,}000$

This approach solves instances with more than queens. The use of aggressive cut generation improves the upper bound on the objective function significantly, though it can be observed that for values of n larger than is not able to deduce the trivial upper bound of n.[17] If we use the following formulation instead of constraint c , this changes:

```
subto c3: forall <i,j> in CxC do
            forall <m,n> in TABU[i,j] do x[i,j] + x[m,n] <= 1;
```

As shown in the table below, the optimal upper bound on the objective function is always found in the root node. This leads to a similar situation as in the integer formulation, i. e., the solution time depends mainly on the time it needs to find the optimal solution. While reducing the number of branch-and-bound nodes evaluated, aggressive cut generation increases the total solution time.

With this approach instances up to queens can be solved. At this point the integer program gets too large to be generated. Even though the presolve routine is able to aggregate the constraints again, Z needs too much memory to generate the . The column labeled *Pres. NZ* lists the number of non-zero entries after the presolve procedure.

Queens	S	Vars	Cons	NZ	Pres. NZ	Root Node	Nodes	Time [s]
16	D	256	12,640	25,280	1,594	16.0	0	<1
32	D	1,024	105,152	210,304	6,060	32.0	58	5
64	D	4,096	857,472	1,714,944	23,970	64.0	110	60
64	C					64.0	30	89
96	D	9,216	2,912,320	5,824,640	53,829	96.0	70	193
96	C					96.0	30	410
96	F					96.0	69	66

17 For the queens instance the optimal solution is found after nodes, but the upper bound is still . .

Finally, we will try the following set packing formulation:

```
subto row: forall <i> in C do
   sum <i,j> in CxC : x[i,j] <= 1;

subto col: forall <j> in C do
   sum <i,j> in CxC : x[i,j] <= 1;

subto diag_row_do: forall <i> in C do
   sum <m,n> in CxC with m - i == n - 1: x[m,n] <= 1;

subto diag_row_up: forall <i> in C do
   sum <m,n> in CxC with m - i == 1 - n: x[m,n] <= 1;

subto diag_col_do: forall <j> in C do
   sum <m,n> in CxC with m - 1 == n - j: x[m,n] <= 1;

subto diag_col_up: forall <j> in C do
   sum <m,n> in CxC with card(C) - m == n - j: x[m,n] <= 1;
```

Here again, the upper bound on the objective function is always optimal. The size of the generated is even smaller than that of the former model after presolve. The results for di erent instances size are shown in the following table:

Queens	S	Vars	Cons	NZ	Root Node	Nodes	Time [s]
64	D	4,096	384	16,512	64.0	0	<1
96	D	9,216	576	37,056	96.0	1680	331
96	C				96.0	1200	338
96	F				96.0	121	15
128	D	16,384	768	65,792	128.0	>7000	>3600
128	F				128.0	309	90

In case of the queens instance with default settings, a solution with queens is found after branch-and-bound nodes, but was not able to find the optimal solution within an hour. From the performance of the Feasible setting it can be presumed that generating cuts is not beneficial for this model.

2.5 Further developments

Z is under active development. The following extensions are planned in the future:

- Improved presolving and postsolving. Up to now only basic presolving algorithms are implemented. Many more are known, e. g., Brearley et al. ( ), Tomlin and Welch (,), Bixby and Wagner ( ), Andersen and Andersen ( ), Fourer and Gay ( ), Savelsbergh ( ), Gondzio ( ), Mészàros and Suhl ( ).
- More extended functions, in particular `vmin`, `vmax` and `vsgn`.
- Direct use of Boolean constraints, like, for example, `subto c1: a and b`, with `a` and `b` being binary variables.
- Additional output formats, for example, -output according to Fourer et al. (b) and an interchange format with (Koch,).
- Automatic LaTeX output of the model like in (Hürlimann,).
- The possibility to specify more than one objective function. This can be useful for example for highly degenerate problems, like those in Chapter . One objective function is an illegitimate perturbed one and the other is the original one.
- More input/data-reading routines for multi-dimensional parameters.
- Incorporation of semi-continuous variables.
- More convenient back-translation of solver output into the original namespace, possibly some kind of report generator.
- A C- to make it possible to embed Z models into programs.

Furthermore limited experiments with extremely large problems (more than million variables) suggest that improved storage schemes for repetitive data like for example lower bounds might be useful.

Chapter 3

Implementing Zimpl

As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.
— Maurice Wilkes discovers debugging,

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.
— Brian W. Kernighan

Beware of bugs in the above code;
I have only proved it correct, not tried it.
— Donald Knuth

In this chapter, we will give an overview on the implementation of Z , describe some features like the set implementation and the extended constraints in more detail, and discuss some of the design decisions taken. We will try not only to describe *what* we have done, but also *how* and *why* we have done it.

3.1 Notes on software engineering

Much of the experience gained by building, maintaining, and porting mathematical software[1] for several years influenced the software engineering practices used when implementing Z .

For a broader view on the subject we would like to refer to the literature, notably the -year anniversary edition of Brooks ( ), which additionally includes as chapter

1 e. g. (Wunderling,), (Martin,), and (Koch,).

the essay "No Silver Bullet" which sums up much of the results on the topic in the last years. For general guidelines on good programming practice we recommend Kernighan and Pike ( ), Bentley (,).

What are the properties good software should have?

- ▶ Correctness
- ▶ Maintainability
- ▶ Extensibility
- ▶ Reusability
- ▶ Portability
- ▶ Efficiency

Correctness is about a program doing what it is intended to do. In a mathematical environment the intention of a program and the algorithms are usually clear and well specified. While any attempts to derive automatic formal proofs of correctness for non-trivial programs failed in practice, there is by means of the *assert* facility some support for runtime checks of preconditions, postconditions, and invariants in C/C++. Other languages as for example Ei el (Meyer,) have far more elaborate mechanisms in this regard. C and C++ both have the property that it is not only very easy for the programmer to introduce errors, but also that these errors are very likely to go unnoticed until disaster strikes, e. g. the program crashes. Examples are out of bound array accesses or even string handling. Errors like these can easily happen in C. C++ on the other hand is a highly complex language, which allows for extremely involved errors. The vast amount of literature on this subject literally speaks volumes (see e. g. Meyers, , , Sutter, , , Dewhurst,). Consequently there is also a wealth of tools available to support the programmer finding these errors. We will introduce some of them in Section . . .

Maintainability is about how easy it is later on to understand the code and fix errors when they are found. The main means to maintainability is to structure programs in a way which allows to conceal the e ect of any change in the program to a small and easy to locate area of code. For example, in a program that only uses global variables and computed goto's it will be very hard to make sure a change will only a ect the right thing. Modular programming aims at concentrating code that conceptually belongs together at a single place, while structured programming strives at making the execution paths of a program easier to understand. Adding to this, object-oriented programming tries to hide the actual data and implementation in objects behind interfaces used to communicate with these objects. In theory this would allow to make sure that no change in the object, which does not change the interface has any influence on the rest of the program.

Extensibility is about how easy further functionality can be incorporated in a program. Programs that are easy to maintain tend also to be extensible, provided the design of the program is broad enough to embrace the extension. In this sense extensibility is mainly a design property.

Reusability is about how easy the more general parts of a program can be reused in other projects. One of the initial promises of object-oriented programming was to considerably increase the amount of reusable code. As became evident, this did not happen. One reason may be the approximately three times higher e ort it needs to produce a reusable component (Brooks, , Chapter). Another question is how much software is reusable in principle. It is not obvious, which software, apart from some general data structures, i. e., lists, trees, etc., and some well-known algorithms, e. g. sorting and some utility functions, is likely to be reused.

Portability is about how easy a program can be compiled and run on a di erent platform. This depends usually on three topics. First, how standard conformant is the code of the program? Second, how standard conformant are the compilers and third, how portable are any libraries and needed utilities like for example parser generators, etc.? The second point is a serious problem in C++ before and since the standardization in . There are interrelations between the first and the second point, as all programming languages have features, which tend to di er more often between compilers than others. Using such features heavily in a program is calling for trouble. The third point is especially important, because it is the most common single reason for insuperable porting problems. Using a library like, for example, the *Microsoft Foundation Classes* makes it virtually impossible to port a program to any non Windows platform.

Efficiency is about how fast the program is running. This is foremost a question of choosing the right algorithms. It is to distinguish between the speed of the program on the intended set of inputs and on a general input of size n. Depending on the runtime complexity of the algorithms employed, a program may well run fast on the intended inputs, but may scale badly in case of much larger inputs. The choice of programming language and the implementation of the algorithms itself seldom imposes a factor of more than , onto the running time of a program.

We tried to address all of the above points in Z . *Assert* statements are extensively used to state preconditions, postconditions, and invariants. One out of six statements in the code is an assertion. It should be noted that assertions also act as a "life" comment in the code. In contrast to real comments, asserts are always[2] correct, since they are verified at runtime.

In order to improve maintainability most data objects in Z are encapsulated and only accessible via their interfaces. This also helps to make the program easier to extend, as for example the complete implementation of sets could be replaced with only minor changes in the rest of the program. The use of parser and lexical analyzer generators makes it easier to extend the language. The encapsulation also made it possible

2 "always" here means "for the checked cases".

to reuse the code dealing with storage and processing of the generated linear integer program from .

System	CPU	OS	Compiler
PC	Pentium-4	Linux/i386 2.6.5	GNU C 3.4.1
PC	Pentium-4	Linux/i386 2.6.5	Intel C 8.0
PC	Athlon	Linux/i386 2.4.21	GNU C 2.95.3
PC	Pentium-4	Windows-XP	GNU C 3.3.1
UP2000+	Alpha EV67	Tru64 5.1B	Compaq C V6.5-207
IBM p690	Power4	AIX 5.1	IBM Visual Age 6
SUN Ultra10	UltraSparc-IIi	Solaris 7	Forte 7 C 5.4
HP 9000/785	PA-RISC 8500	HP-UX 11.00	HP C B.11.11.04
Onyx 3000	MIPS R12000	IRIX 6.5	MIPSpro 7.41

Table . : Platforms Z compiles and runs on

Regarding portability Table . lists all platforms on which we have compiled and tested Z . Additionally users have reported successful builds on Linux/ and - / . It should be noted that the Windows- version was built using a cross-compiler on Linux and the MinGW[3] toolkit.

To make Z e cient we tried to use the proper algorithms and, as we will see later, succeeded in considerably improving the running time of Z on larger inputs. The use of C as programming language assures that, using the same algorithms, not much can be gained in terms of execution speed by the use of another language (Kernighan and Pike, , page).

3.2 Zimpl overview

Z is an interpreter. Each statement is first parsed and translated into a parse tree consisting of code nodes. Then the resulting code tree for the statement is traversed, thereby executing the statement. The Z main-loop works as follows:

```
while(input available)
begin
   read_next_statement
   parse_statement_into_code_tree
   execute_code_tree
end
```

An alternative to this tree walking evaluation is the implementation of a virtual machine, like for example a stack machine (Kernighan and Pike,). But since there is no need to store the parsed (compiled) input, there seem to be no particular advantages of taking the other approach.

3 http://www.mingw.org

Each node in the code tree has the following structure:

```
typedef struct statement    Stmt;
typedef enum code_type      CodeType;
typedef union code_value    CodeValue;
typedef struct code_node    CodeNode;
typedef CodeNode*        (*Inst)(CodeNode* self);

struct code_node
{
   Inst         eval;
   CodeType     type;
   CodeValue    value;
   CodeNode*    child[MAX_CHILDS];
   const Stmt*  stmt;
   int          column;
};
```

`Inst` is a pointer to a C-function that takes this code node as an argument. The function evaluates to the `value` of the node, which is of type `type`. `child` contains the arguments of the function, which in turn are also code nodes. Finally `statement` and `column` identify the position in the input line this code node stems from. This is used in case of errors to localize the error in the input.

If `inst` is executed, it will fill `value` as a result. To obtain the arguments needed for `inst`, all valid code nodes in `child` will be executed. This leads to a recursion that traverses the code tree after the execution of the root node and sets all `value` fields in the tree. The code tree resulting from the input `param eps:=5*7;`. is drawn in Figure . . The `inst` functions often have side e ects, e. g., they define sets, parameters, or constraints that persist after the execution of the statement has finished and are not part of `result`.

3.3 The parser

We will not give an introduction into formal language parsing, but only note a few details about the Z grammar which by all standards is quite simple (to parse). The Z grammar itself is listed in Appendix B. on page .

3.3.1 BISON as parser generator

The input is broken into tokens with a lexical analyzer generated by [4], a replacement of the tool (Lesk,). The resulting token stream is fed to the parser which is generated from a grammar description with [5], an upward compatible replacement of ("Yet Another Compiler Compiler", Johnson,). is a general-purpose parser generator that converts a grammar description for a ( )[6] context-free grammar into a C program to parse that grammar.

4 http://sourceforge.net/projects/lex
5 http://www.gnu.org/software/bison
6 One token Look Ahead Left Recursive, see for example Aho et al. ( ), Holub ( ).

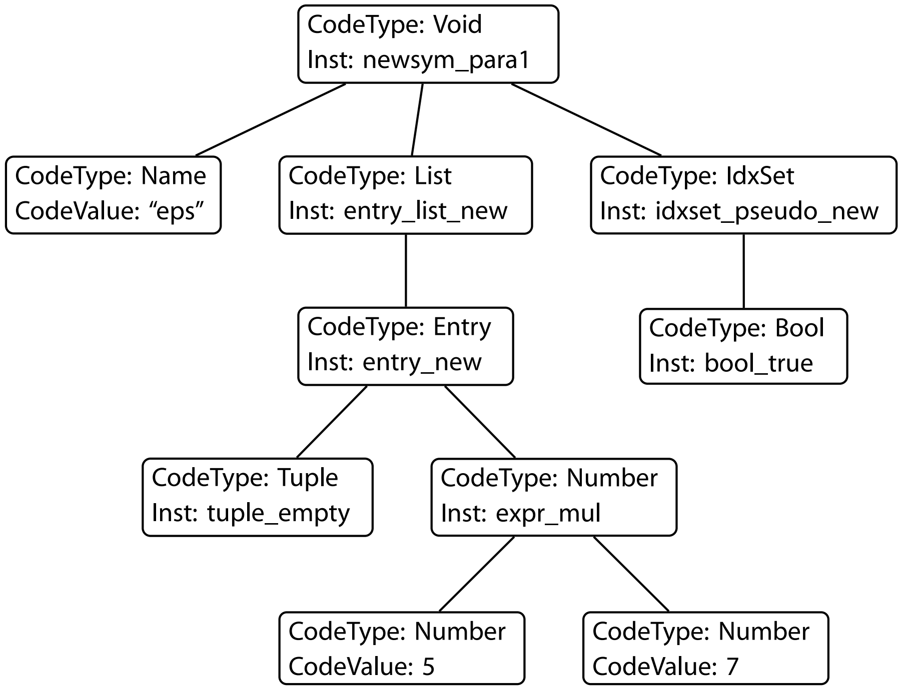


Figure . : Tree of code-nodes

( ) means apart from other things that the parser only looks one token ahead, when deciding what to do. This means the following grammar, describing a declaration with two optional arguments, is ambiguous to the parser:

```
%token DECL NUMB STRG
%%
stmt : DECL STRG par1 par2;
par1 : /* empty */ | ',' NUMB ;
par2 : /* empty */ | ',' STRG ;
```

After getting a comma the parser cannot decide whether it belongs to `par1` or `par2`, because the parser lacks the lookahead to see if a NUMB or a STRG is next.

The solution to these kind of problems are either to reformulate the grammar, e. g., to enumerate all combinations, or handle the problem later in the parse tree, e. g., drop the di erentiation between NUMB and STRG at this point in the grammar and handle the di erence when evaluating the tree.

By the standards of computer science or are rather ancient tools. They have even been standardized as part of . (.). There are newer tools to generate parsers, like for example [7], [8], and [9]. even allows

7 http://dparser.sourceforge.net
8 http://www.antlr.org
9 http://www.hwaci.com/sw/lemon

ambiguous grammars. The reason why we still used to implement Z is that is a very mature and well-known program, i. e., has few bugs, is freely available on practically all platforms[10], is standard conformant, and will remain available and maintained for the foreseeable future. Additionally, there is a decent amount of literature on how to use it, like Schreiner and Friedman ( ), Levine et al. ( ). Since Z has no elaborate demands on its grammar, seemed to be the right tool. But up to now there is no indication whether this choice was fortunate or not.

3.3.2 Reserved words

Every language designer has to decide how keywords should be handled. There are basically three possibilities:

- Use reserved words, like e. g. C. This makes parsing and error detection easy, but keywords can clash with user chosen names.
- Use separate namespaces for variables, sets, and parameters, for example, by prefixing them with $, #, and &, like e. g. . Again parsing and error detection is easy, but the code looks ugly.
- Do not use reserved words and decide from the grammar what is a keyword, like e. g. / . In this case there are no restrictions on the names the user chooses, but parsing and error detection gets a lot more di cult. Sometimes even for the user, as `IF IF==THEN THEN THEN ELSE ELSE END` would be legal.

We decided to reserve the names of the keywords, but it might be an idea to use a prefix for built-in functions like 'abs' because they make the majority of the reserved words (Z has reserved words, of them stemming from built-in functions). Additionally, there is a number of reserved words that can be easily distinguished from names just by their position in the input. `set`, for example, is only allowed at the start of a line, a position where never any user chosen name is feasible.

3.3.3 Type checking

An important decision is whether the parser checks the types of operands and arguments or if this is deferred until the value is actually needed within a function.

Type checking within the parser

- All functions can assume that their arguments are of the expected type.
- All cases have to be explicitly handled in the parser.
- It is only possible to decide on the type not on the value of a token, e. g. division by zero cannot be handled in this way.
- The parser performs all the error reporting, i. e., the localization of the errors is rather accurate, but the parser cannot report much about the error itself.

10 The portability of the generated code is more important than the portability of the generator itself.

Type checking at runtime

- The parser gets simpler and more orthogonal.
- All information on type and value is available.
- All cases have to be handled in every function at execution time.
- It is more di cult to precisely locate the position of an error.

In Z a mixed approach is used. Constant numbers and strings are the only types not distinguished within the parser. All functions have to make sure that they either get what they need, or that they perform the right operation depending on the types of their arguments. Since only two rather distinct types are involved, the case handling within the functions remains simple. On the other hand, the parser gets considerably smaller and simpler, especially for the handling of components of tuples, which can be of either type.

The Z parser is very restrictive and will reject most invalid constructions. To make it possible for the parser to detect wrong argument and operator types, Boolean expressions, set expression, and especially all expressions involving decision variables are handled separately in the parser. This also simplifies and speeds up the argument processing within the functions. As can be seen in the grammar (page), distinguishing between expressions with and without decision variables leads to a slightly involved construction for variable expressions (`vexpr`). This makes it rather di cult to assure the correctness of the parser, especially regarding precedence between operators.

Symbol table lookup

One problem with letting the parser handle type checking is that while tokenizing the input, the type for named entities like parameters has to be known.

In Z it is only valid to use named entities like a set or a parameter after they have been defined. This allows to lookup any names already in the lexical analyzer and to determine their type. As an extra benefit, any name has only to be looked up once in the analyzer and not again when executing the function.

3.4 Implementation of sets

Since all repetitive operations in Z are done over sets, the implementation of sets has a severe impact on the performance. A set is a collection of objects. The only required property of a set is the possibility to ask, whether some object is a member of the set or not. This has an important consequence regarding the implementation of a set data structure: Each element can only be in a set once. For the implementation this means in order to add an object to a set, we have to know whether it is already a member of the set.

These requirements fit very well with the properties of the *hash table* data structure, see for example Knuth (b, chapter .) or Aho and Ullman (, chapter .). The average time needed to insert or to find an element in a hash table is $\mathcal{O}(1+n/B)$, where n is the number of elements in the table and B is the size of the table. This shows immediately the biggest disadvantage of a hash table, namely that its size B has to be set in advance and its performance drops as n approaches and exceeds B. On the other hand, if n stays very small compared to B, a lot of memory is wasted.[11]

Internally, sets in Z are ordered sets of n-tuples, i. e., each member of a set has the same number of components and a distinct index. We denote the index $i \in \{1, \dots, |A|\}$ of an element $a \in A$ of an ordered set A by $\sigma(a) = i$. For a n-tuple t we denote by $t_i, i \in \{1, \dots, n\}$ the i-th component of t. Given a set A of n-tuples, an ordered set of indices $K \subseteq \{k \mid k \in \{1, \dots, n\}\}$ and a $|K|$-tuple p, we define a *slice*(A, K, p) as $\{a \in A \mid a_k = p_{\sigma(k)}$ for all $k \in K\}$.

To store the cross product of two sets we have basically two possibilities: Either we factor the product, i. e., we explicitly generate and store all members of the cross product, or we record the two operand sets and generate the elements on demand. The latter approach has two advantages: It needs less memory and it allows to speed up the iteration over a slice of the set.

Sets within Z are implemented as an abstract virtual base class[12] with the following data members:

```
struct set_head {
   int        refc;     /* reference count      */
   int        dim;      /* dimension of tuples  */
   int        members;  /* number of set members */
   SetType    type;     /* Type of the set      */
};
```

Since within a Z program once created sets never change, it is possible to implement the copy operation on sets by the use of a reference counter.

At present, four di erent derivate implementations for sets are used. Two types of singleton sets, namely *list-sets* which store singletons, either in a hash table or as a list[13], and *range-sets*, which store sets of numbers described by *begin*, *end*, and *interval*. More precisely, a range-set for parameters $b, e, i \in \mathbb{Z}$ is defined as *range*$(b, e, i) = \{x \in \mathbb{Z} \mid x = b + ni, n \in \mathbb{N}_0, b \leqslant x \leqslant e\}$.

```
struct set_list {
   SetHead  head;    /* head.dim == 1                          */
   Elem**   member;  /* head.members holds the number of elements */
   Hash*    hash;    /* Hash table for members                 */
};
```

11 There exist some improved algorithms which overcome at least part of these problems, see, for example, Pagh and Rodler ( ).
12 Since Z is programmed in C this is implemented "by hand".
13 The memory overhead of hash tables starts to be a problem if excessive numbers of sets are created, e. g., due to the construction of a powerset. In these cases a more condensed storage is needed, sacrificing some performance when looking for an element. For this reason our implementation does not create hash tables for sets which have less than twelve elements. Sequential search is employed to find an element in these cases.

```
struct set_range {
   SetHead head;    /* head.dim == 1              */
   int     begin;   /* First number               */
   int     end;     /* The last number is <= end  */
   int     step;    /* Interval                   */
};
```

Further, we have two types of composite sets, namely *product-sets* which represent the cross-product of two sets, and *multi-sets* which represent a subset of the cross-product of n-list-sets.

```
struct set_prod {
   SetHead head;    /* head.dim > 1 */
   Set*    set_a;   /* A from A * B */
   Set*    set_b;   /* B from A * B */
};

struct set_multi {
   SetHead head;    /* head.dim > 1  */
   Set**   set;     /* head.dim holds number of involved sets */
   int*    subset;  /* List members, size head.members * head.dim */
   int**   order;   /* All orders, size head.dim, head.members */
};
```

For a product-set not more than references to the two involved sets, which can be of any type, are stored. Since multi-sets reference only singleton list-sets, `head.dim` equals the number of involved list-sets. `subset` is a list holding for each component of each present member the index of the component in the list-set. `order` holds indices of the subset list sorted by each component. While not much of an improvement from a theoretical point of view, practically this allows us a much faster computation of slices.

Initially all sets are built from elements that are supplied as part of the data. This can happen in three possible ways:

- The data is a list of singleton elements. In this case a list-set is created to store the elements.
- The data is a list of n-tuples ($n > 1$). In this case n list-sets are built from the n projections of the elements to a single component. Next a *multi-set* is constructed to index the available elements by referencing the list-sets.
- The data is given as a range. In this case a *range-set* storing *begin*, *end*, and *interval* is created.

After initial sets are built from data, further sets can result from various set operations. Building the cross-product of two sets is done by creating a cross-set, which is, regarding both running time and memory requirements, an $\mathcal{O}(1)$ operation. The rest of the set operators like union or intersection are implemented by building a list of the resulting elements and then generating an appropriate *multi-set*.

3.5 Implementation of hashing

Hashing is used in Z to look up singleton elements of sets and the names of variables, parameters and sets. We now give some details and statistics on the hash functions employed.

A hash function $h(A) \to \{0, \ldots, N-1\}$ is a mapping of a set A into a bounded interval of integers. Usually $|A| \gg N$ and h is not injective. The case $h(a) = h(b)$ for $a \neq b$, is called a *collision*. According to Knuth (b, page) a good hash function should satisfy two requirements: Its computation should be very fast and it should minimize collisions. The first property is machine-dependent, and the second property is data-dependent.

We compare five hash functions for strings. As a test-set we use variable names as they are typically generated within Z : `x#1`,...,`x#n`. The implementations have always the same function hull:

```
1 unsigned int str_hash1(const char* s)
2 {
3    unsigned int sum = 0, i;
4
5    /* insert computing loop here */
6
7    return sum % TABLE_SIZE;
8 }
```

The following hash algorithms were inserted at line in the above routine:

) `for(i = 0; i < strlen(s), i++) sum = sum + s[i];`
This is sometimes found in textbooks as a "simple" hash function for strings.

) `for(i = 0; i < strlen(s), i++) sum = sum * 32 + s[i];`
This is the hash function given in Sedgewick (, page).

) `for(i = 0; i < strlen(s), i++) sum = sum * 31 + s[i];`
This one can be found in Kernighan and Pike (, page).

) `for(i = 0; i < strlen(s), i++) sum = DISPERSE(sum + s[i]);`
In this case a linear congruence generator (see, e. g., Press et al.,) is used. `DISPERSE(x)` is defined as `1664525U * x + 1013904223U`.

) `for(i = strlen(s) - 1; i >= 0; i--) sum = DISPERSE(sum + s[i]);`
This is equal to the former, but the characters are processed in reverse order.

In case of a collision, the entry is added to a linked list of entries with the same hash value. This is called *separate chaining* and has the advantage of not limiting the number of entries that fit into the hash table.

We tested the performance for $n = 10^3, 10^4, 10^5, 10^6$, and 10^7. The size of the hash table itself (`TABLE_SIZE`) was always set to , , . The results are listed in Table . . *Average chain length* is the average length of non-empty chains. *Maximum chain length* gives the length of the longest chain built from the input. Note that an average chain length of . for $n = 10^7$ entries is about , , / , , which indicates that the distribution of hash values is symmetric.

$n =$	1,000	10,000	100,000	1,000,000	10,000,000
Algorithm 1 (simple sum)					
Average chain length	18.5	111.1	740.733	5,291	40,650.4
Maximum chain length	70	615	5,520	50,412	468,448
Algorithm 2 (spread 32)					
Average chain length	1	1	1.1238	1.859	10.0771
Maximum chain length	1	1	2	7	31
Algorithm 3 (spread 31)					
Average chain length	1	1	1.00422	1.4038	9.99997
Maximum chain length	1	1	2	4	20
Algorithm 4 (randomize)					
Average chain length	1	1.00807	1.04945	1.57894	9.99997
Maximum chain length	1	2	3	6	23
Algorithm 5 (reverse randomize)					
Average chain length	1	1	1.0111	1.44194	9.99997
Maximum chain length	1	1	2	6	20

Table . : Performance of hash functions (table-size = , ,)

We also tested the , word dictionary from /usr/share/dict/words. The results are basically the same: ( ) and ( ) give similar good performance, and ( ) works very badly.

Up to this test, we employed algorithm ( ) within Z , but this changed as a result of the test, since evidently algorithm ( ) performs significantly better and is faster. In the current version of Z the algorithm to decide the size of a hash table works as follows: From a list of primes, find the first prime greater than two times the maximum number of entries in the hash table. The test revealed that for algorithms ( ), ( ), and ( ) the chain length for % of the non-empty chains is three or less if the number of entries equals the size of the hash table. It seems therefore appropriate to no longer double the anticipated number of entries before deciding the size of the table.

3.6 Arithmetic

In this section we give some information about floating-point and rational arithmetic to make it easier to understand the benefits and drawbacks resulting from using rational arithmetic within Z .

3.6.1 Floating-point arithmetic

The general representation of a floating-point number with p digits is:

$$\pm(d_0 + d_1\beta^{-1} + \ldots + d_{p-1}\beta^{-(p-1)})\beta^e$$

where $\beta \in \mathbb{N}$ is the basis, $e \in \mathbb{Z}$ is the exponent, and $0 \leqslant d_i < \beta$ for $0 \leqslant i < p$ are the digits. If $d_0 \neq 0$ we call the number *normalized*. Today's computers usually use binary floating-point arithmetic according to the standard.[14] This means, double precision numbers, for example, are bits wide using the following layout:

```
S EEEEEEEEEEE DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
0 1        11 12                                                63
```

where S is the sign of the number, the E's are the exponent and the D's contain the digits of the mantissa. Obviously $\beta = 2$ for a binary number. Storing the number normalized, gains one digit, since we know that the first digit will be a one, so we do not need to store it. This means we have $p = 53$. The disadvantage of normalized numbers is that they cannot represent zero. So the exponents zero and , are reserved to indicate "special" values like zero and infinity. We get the value of a binary floating-point number by building

$$v = (-1)^S 2^{(E-1023)}(1.D)$$

where 1.D represents the binary number created by prefixing the D's with an implicit leading one and a binary point.

Note that numbers like for example . have no finite representation as binary-point number. Rounding can result from arithmetic operations, e. g., / = . , which is represented as . . Since the size of the exponent is limited, overflows and underflows can happen as the result of arithmetic operations. Absorption is also possible, i. e., $a + b = a$ for $b \neq 0$. Finally, the subtraction between very similar operands can lead to nearly random results.

Table . lists the parameters for typical floating-point numbers. The last column gives the smallest number ϵ for which $1 + \epsilon > 1$ is true.

Precision	Bits	Bytes	p	e	Max/Min	$1 + \epsilon > 1$
Single	32	4	24	8	$\approx 10^{\pm 38}$	$\approx 1.1920 \cdot 10^{-7}$
Double	64	8	53	11	$\approx 10^{\pm 308}$	$\approx 2.2204 \cdot 10^{-16}$
Extended	80	6	64	15	$\approx 10^{\pm 4932}$	$\approx 1.0842 \cdot 10^{-19}$
Quad	128	16	113	15	$\approx 10^{\pm 4932}$	$\approx 1.9259 \cdot 10^{-34}$

Table . : Standard floating-point format parameters

The program shown in Figure . can be used to measure the number of mantissa digits p for di erent data types. One would expect identical output for lines and , and for and , and for and . Table . shows the reality. The reason for these seemingly strange results is the following: The length of the floating-point registers of the and - s di er from the length of the corresponding memory storage locations for the respective data types. - 's, for example, have bit registers, i. e., use extended precision internally for all computations. Whenever the contents of a register has to be written to memory, it is truncated to the specified precision.

14 http://standards.ieee.org

```
#include <stdio.h>

float       sf[256];
double      sd[256];
long double sl[256];

void compute_mantissa_bits(void)
{
   float       af, ef;
   double      ad, ed;
   long double al, el;
   int         n;

   af = ef = 1.0; n = 0;
   do { ef /= 2.0; sf[n] = af + ef; n++; } while(sf[n - 1] > af);
   printf("f(%3d) p=%d %.16e\n", sizeof(af)*8, n, 2.0*ef);

   af = ef = 1.0; n  = 0;
   do { ef /= 2.0; n++; } while(af + ef > af);
   printf("f(%3d) p=%d %.16e\n", sizeof(af)*8, n, 2.0*ef);

   ad = ed = 1.0; n = 0;
   do { ed /= 2.0; sd[n] = ad + ed; n++; } while(sd[n - 1] > ad);
   printf("d(%3d) p=%d %.16e\n", sizeof(ad)*8, n, 2.0*ed);

   ad = ed = 1.0; n  = 0;
   do { ed /= 2.0; n++; } while(ad + ed > ad);
   printf("d(%3d) p=%d %.16e\n", sizeof(ad)*8, n, 2.0*ed);

   al = el = 1.0L; n = 0;
   do { el /= 2.0L; sl[n] = al + el; n++; } while(sl[n - 1] > al);
   printf("l(%3d) p=%d %.16Le\n", sizeof(al)*8, n, 2.0L*el);

   al = el = 1.0L; n  = 0;
   do { el /= 2.0L; n++; } while(al + el > al);
   printf("l(%3d) p=%d %.16Le\n", sizeof(al)*8, n, 2.0L*el);
}
```

Figure . : C-function to determine floating-point precision

Which registers are written to memory at which point of the computation is entirely[15] dependent on the compiler.

The results in the last two rows of Table . highlight what is generally true for floating-point calculations, namely that the order in which operations are carried out can change the result of a floating-point calculation. This means that floating-point arithmetic is neither associative nor distributive. Two mathematically equivalent formulas may not produce the same numerical output, and one may be substantially more accurate than the other.

15 Of course, knowing the restrictions of the architectures, it is possible to derive programs like the one in Figure . that forces the compilers to generate code that writes the register contents to memory.

System	float		double		long double	
Source line	16	20	24	28	32	36
Alpha EV67	24	24	53	53	113	113
UltraSPARC-IIi	24	24	53	53	113	113
PA-RISC 8500	24	24	53	53	113	113
POWER4	24	53	53	53	53	53
IA-32	24	64	53	64	64	64

Table . : Mantissa bits p in floating-point computations

More details about floating-point numbers can be found among others in "What every computer scientist should know about floating-point arithmetic" (Goldberg,), in Knuth (a), and in the Internet.[16]

3.6.2 Rational arithmetic

Unlimited precision rational arithmetic has two severe inherent drawbacks:

- "Unlimited precision" also means unlimited time and space requirements.
- Rational arithmetic cannot compute non-rational functions, e. g. square roots.

Apart from these principle problems, rational arithmetic needs considerable more time and space even for limited precision computations. When verifying optimal simplex bases with (Koch,), , bits were needed to store the objective function value for the linear program *maros-r7* from the (Gay,). This corresponds to more than , decimal digits (nominator and denominator together). Performing a single factorization of the basis and solving the equation system took approximately between and times longer than doing a similar operation in double precision floating-point arithmetic.

The reason for this slowdown is not only the increased size of the numbers involved: Since each number is stored as a numerator/denominator pair common factors have to be handled. According to the manual it is believed that casting out common factors at each stage of a calculation is best in general. A is an $\mathcal{O}(n^2)$ operation, so it's better to do a few small ones immediately than to delay and have to perform a on a big number later.

Within Z , the *Multiple Precision Arithmetic Library* ( ) is used for all rational arithmetic. For more information on how to implement rational arithmetic see, for example, the website[17] or Knuth (a). If the computation of non-rational functions is requested in a Z program, the operations are performed with double precision floating-point arithmetic.

16 e. g. http://en.wikipedia.org/wiki/Floating_point,
http://babbage.cs.qc.edu/courses/cs341/IEEE-754references.html

17 http://www.swox.com/gmp

3.7 Extended modeling

In this section we describe how what we call *extended constraints and functions* are modeled. Information on this topic can be found, for example, in Williams and Brailsford ( ), Plastria ( ), or at the website[18].

Given a bounded integer variable $l_x \leqslant x \leqslant u_x, l_x, x, u_x \in \mathbb{Z}$, we introduce two additional binary variables b^+ and b^- as indicators for whether x is positive or negative, i. e., $b^+ = 1$ if and only if $x > 0$ and $b^- = 1$ if and only if $x < 0$. In case of $x = 0$, both b^+ and b^- equals zero. Further we introduce two non-negative variables x^+ and x^- which hold the positive and negative portion of x. We can formulate this as an integer program:

$$\begin{array}{rcccl} & & x^+ - x^- & = & x \\ b^+ & \leqslant & x^+ & \leqslant & \max(0, u_x)b^+ \\ b^- & \leqslant & x^- & \leqslant & |\min(0, l_x)|b^- \\ & & b^+ + b^- & \leqslant & 1 \\ & & b^+, b^- & \in & \{0, 1\} \end{array} \qquad (\ . \ )$$

Theorem 1. *The polyhedron described by the linear relaxation of system* (.) *has only integral vertices.*

Proof. For fixed l_x and u_x we can write the relaxation of (.) as

$$\begin{array}{lccccll} (1) & x^+ & & -\lambda b^+ & & \leqslant & 0 \\ (2) & x^+ & & -b^+ & & \geqslant & 0 \\ (3) & & & b^+ & & \leqslant & 1 \\ (4) & & x^- & & -\mu b^- & \leqslant & 0 \\ (5) & & x^- & & -b^- & \geqslant & 0 \\ (6) & & & & b^- & \leqslant & 1 \\ (7) & & & b^+ & +b^- & \leqslant & 1 \end{array}$$

with $\lambda, \mu \in \mathbb{N}_0$. For $\lambda = 0$ ( )-( ) result in $x^+ = b^+ = 0$ and correspondingly for $\mu = 0$ ( )-( ) become $x^- = b^- = 0$. For $\lambda = 1$ ( )-( ) degenerate to $b^+ = x^+ \leqslant 1$ and for $\mu = 1$ the same happens for ( )-( ). For $\lambda \geqslant 2$ the polyhedron described by ( )-( ) has only vertices at $(x^+, b^+) = (0, 0)$, $(1, 1)$, and $(\lambda, 1)$. Figure . shows an example.

Inequalities ( )-( ) are similar for $\mu \geqslant 2$. The only connection between the two systems is through ( ). We need four equalities to describe a vertex. Taking two from ( )-( ) and two from ( )-( ) will lead to an integral point because there is no connection. Any combination of ( ) together with ( ) and ( ) has obviously no point of intersection. Given ( ) together with either ( ) or ( ), (b^+, b^-) is set to either $(1, 0)$ or $(0, 1)$ and x^+ and x^- have to be integral. This leaves ( ) with three choices from ( ), ( ), ( ), and ( ). For $\lambda, \mu \geqslant 2$ any combination of ( ) and ( ), or ( ) and ( ) has only point $(x^+, b^+) = (0, 0)$ or $(x^-, b^-) = (0, 0)$, respectively, as intersection, forcing integral values for the other variables. □

18 http://www.gams.com/modlib/libhtml/absmip.htm

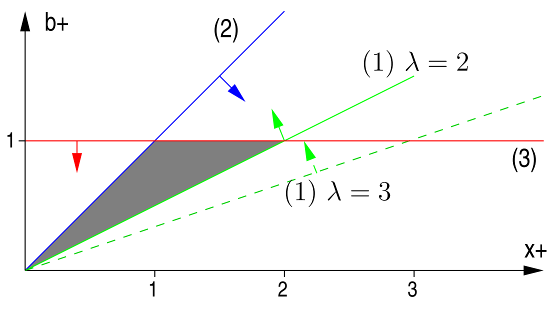


Figure . : Polyhedron defined by ( )-( )

Using system (.) the following functions and relations can be modeled using $x = v - w, v, w \in \mathbb{Z}$ with $l_x = l_v - u_w$ and $u_x = u_v - l_w$ whenever two operands are involved:

$$
\begin{array}{lcl}
\mathrm{abs}(x) & = & x^+ + x^- \\
\mathrm{sgn}(x) & = & b^+ - b^- \\
\min(v, w) & = & w - x^- \\
\max(v, w) & = & x^+ + w \\
v \neq w & \Leftrightarrow & b^+ + b^- = 1 \\
v = w & \Leftrightarrow & b^+ + b^- = 0 \\
v \leqslant w & \Leftrightarrow & b^+ = 0 \\
v < w & \Leftrightarrow & b^- = 1 \\
v \geqslant w & \Leftrightarrow & b^- = 0 \\
v > w & \Leftrightarrow & b^+ = 1
\end{array}
$$

As an example we will show the proof of the correctness of *sign* function $\mathrm{sgn}(x) = b^+ - b^-$. Proofs for the other functions and relations are similar.

$$
\begin{array}{lrlll}
\mathrm{sgn}(x) = & 1 & \Leftrightarrow x > 0 \Rightarrow x^+ > 0 \Rightarrow b^+ = 1 \Rightarrow b^- = 0, & \text{and } b^+ - b^- = 1 \\
\mathrm{sgn}(x) = & -1 & \Leftrightarrow x < 0 \Rightarrow x^- > 0 \Rightarrow b^- = 1 \Rightarrow b^+ = 0, & \text{and } b^+ - b^- = -1 \\
\mathrm{sgn}(x) = & 0 & \Leftrightarrow x = 0 \Rightarrow x^+ = x^- = 0 \Rightarrow b^+ = b^- = 0, & \text{and } b^+ - b^- = 0
\end{array}
$$

Variations

For the functions min, max, and abs where neither b^+ nor b^- is part of the result, one of the two variables can be replaced by the complement of the other. Also the restrictions $b^+ \leqslant x^+$ and $b^- \leqslant x^-$ can be dropped from (.).

If x is non-negative, system (.) can be simplified to $b^+ \leqslant x \leqslant u_x b^+$ with $b^+ \in \{0, 1\}$, $x^+ = x$, $x^- = 0$, and $b^- = 0$. The same arguments apply if x is non-positive.

As mentioned earlier, nearly all solvers use feasibility, optimality, and integrality tolerances. In practice, it might be necessary to define a zero tolerance, equal or greater than the integrality tolerance of the solver and augment (.) accordingly.

3.7.1 Boolean operations on binary variables

In the following we show how to compute a binary variable a as the result of a Boolean operation on two binary variables b and c. Note that the polyhedra defined by the linear relaxations of these inequalities have solely integral vertices, as can be checked with [19].

$a = b$ **and** c

$$\begin{aligned} a - b &\leqslant 0 \\ a - c &\leqslant 0 \\ a - b - c &\geqslant -1 \end{aligned} \qquad (\; . \;)$$

$a = b$ **or** c

$$\begin{aligned} a - b &\geqslant 0 \\ a - c &\geqslant 0 \\ a - b - c &\leqslant 0 \end{aligned}$$

$a =$ **not** b

$$a + b = 1$$

$a = b$ **xor** c

$$\begin{aligned} a - b - c &\leqslant 0 \\ a - b + c &\geqslant 0 \\ a + b - c &\geqslant 0 \\ a + b + c &\leqslant 2 \end{aligned}$$

or alternatively by introducing an additional binary variable d:

$$a + b + c = 2d$$

Note, that the polyhedron of the linear relaxation of this alternative formulation has non-integral vertices, e. g., $a = b = 0$, $c = 1$, and $d = 0.5$.

3.7.2 Conditional execution

Given a Boolean variable r and a bounded constraint $x = \sum a_i x_i \leqslant b$ with

$$U = \sum_{a_i<0} a_i l_{x_i} + \sum_{a_i>0} a_i u_{x_i}$$

We want the constraint to be active if and only if $r = 1$. We can model this by defining a "big" $M = U - b$ and writing:

$$x + Mr \leqslant U \qquad (\; . \;)$$

19 http://www.zib.de/Optimization/Software/Porta

Proof. In case $r = 1$ (.) becomes $x + U - b \leqslant U$ which is equivalent to $x \leqslant b$. In case $r = 0$ we obtain $x \leqslant U$ which is true by definition. □

The formulation for $x = \sum a_i x_i \geqslant b$ with $L = \sum_{a_i<0} a_i u_{x_i} + \sum_{a_i>0} a_i l_{x_i}$ can be obtained from (.) by exchanging U for L and changing less-or-equal to greater-or-equal. To model conditional equality constraints, both a less-or-equal and a greater-or-equal formulation have to be used.

3.8 The history of Zimpl

After we have seen how specific components are implemented, we want to give a short history of the Z development. Table . lists all Z -versions so far, their release dates, the total number of code lines (including comments and empty lines) and the most important change of the respective version. As can be seen from the table, the size of the source code has roughly doubled since the first release. It is also evident that Z is not a big project by any standards. But as we will see in the next sections, quality assurance and testing even for a project of this moderate size is an arduous task.

Version	Date	LOC	Ex1 [s]	Ex2 [s]	
1.00	Oct 2001	10,748	34	2,713	Initial release
1.01	Oct 2001	10,396	34	2,647	Bug fixes
1.02	Jul 2002	10,887	34	2,687	Constraint attributes
1.03	Jul 2002	11,423	50	4,114	Nested forall
1.04	Oct 2002	11,986	50	4,145	General enhancements
1.05	Mar 2003	12,166	44	4,127	Indexed sets, powersets
2.00	Sep 2003	17,578	67	6,093	Rational arithmetic
2.01	Oct 2003	19,287	71	6,093	Extended functions
2.02	May 2004	22,414	2	12	New set implementation

Table . : Comparison of Z versions.

Columns *Ex1* and *Ex2* give the result of a comparison of the time needed by the different Z versions to process a slightly modified[20] version of the set covering model listed in Appendix C. on page . The most important constraint in the model is

```
subto c1: forall <p> in P do sum <s,p> in SP : x[s] >= 1;.
```

The sizes of the sets are for *ex1*: |P|= , and |SP| = , and for *ex2*: |P|= , and |SP|= , . The slowdown from the use of rational arithmetic evident in the table is due to the increased setup and storage requirements, since no arithmetic is performed. Obviously other changes done since version . have had a similar impact on processing time. The most visible improvement is the new set implementation described in this chapter. For all previous versions the execution of *ex2* took about times longer than the execution of *ex1*. With version . the factor is down to six, which is about the di erence in size of the two examples.

20 The changes were necessary to make it compatible with all versions.

3.9 Quality assurance and testing

Z has now been maintained and extended for about four years. Coming back to our initial questions about correctness and maintainability we will describe how we tried to make sure that new features work as anticipated and that the total number of errors in the code is strictly monotonously decreasing.[21]

The call graph of a program run is a directed graph obtained by making a node for each function and connecting two nodes by an arc, if the first function calls the second. A directed path between two nodes in the call graph represents a path of control flow in the program that starts at the first function and reaches the second one. Note that there is no representation for iteration in the call graph, while recursion is represented by a directed circle in the graph.

Figure . shows about one third of the call graph generated by a run of the n-queens problem in Section . . . It is immediately clear from the picture that proving the correctness of the program is a di cult to nearly impossible task. But even though, we strive at least to minimize the number of errors. A key issue in this regard is to make sure that fixing bugs does not introduce new ones.

3.9.1 Regression tests

One of the most important tools in maintaining a program are (automated) regression tests. In fact there is a trend to actually start programming with building a test-suite for the intended features (Beck, ,). Having a test-suite the program has to pass diminishes the chances that a modification breaks working features unnoticed. Since the tests have to be extended for each new feature, those are tested as well. If a bug is found in a program, the tests used to reproduce the problem can be added to the test-suite to make sure that the bug cannot reappear unnoticed.

Test coverage

How do we know if we have a su cient amount of tests? An easy measure is the percentage of statements the test-suite covers, i. e., the percentage of all statements that get executed during the tests. Tools like for example [22], ++[23], or - [24] can do this for C/C++ code. It was a revealing exercise to contrive test-cases for all error messages Z can produce. As it turned out, some errors could not occur, while other error conditions did not produce a message. Currently the Z test-suite consists of tests for error and warning messages and eleven bigger feature tests. As we will see later on, in Table . , we reach % of the statements. There are several reasons for the

21 Tanenbaum (, page) about / : *. . . and contained thousands upon thousands of bugs, which necessitated a continuous stream of new releases in an attempt to correct them. Each new release fixed some bugs and introduced new ones, so the number of bugs probably remained constant in time.*

22 http://gcc.gnu.org

23 http://www.parasoft.com

24 http://www.ibm.com/software/rational

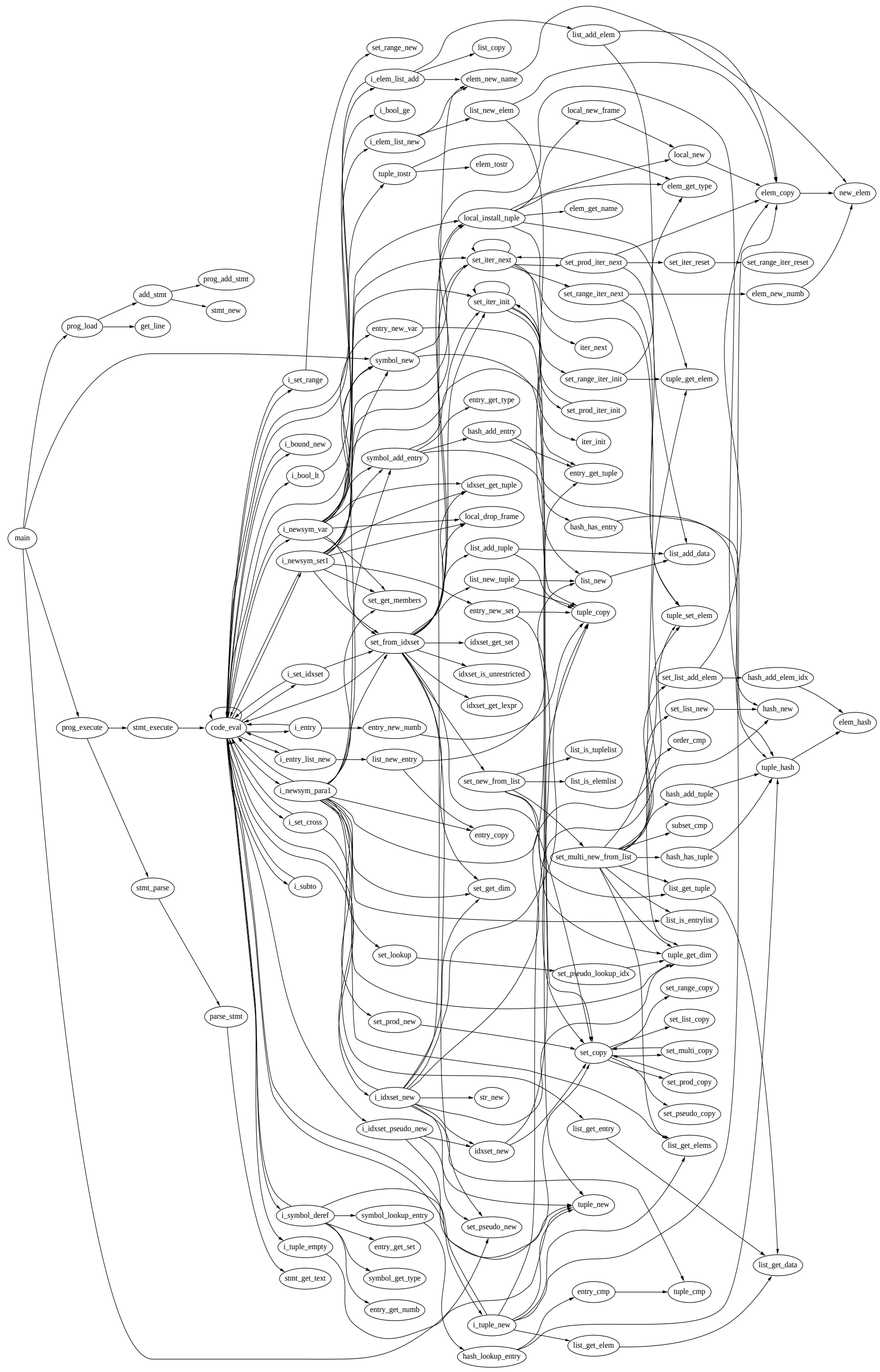


Figure . : Part of the call graph from the n-queens problem in Section . .

missing %. First we have some error conditions like "out of memory" which are difficult to test. Then there is a number of small functions that are not executed at all by the tests. Several of them are not part of the regular program but testing and debugging aids. And finally there are some functions for which testing should be improved.

3.9.2 Software metrics

Besides test coverage, it would be useful to have measures or "metrics" which help to locate areas in the program which are di cult to test and maintain, and which are likely to cause trouble. Practically all books on software metrics start with the famous quote from Lord Kelvin, given in

> *When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind.*

This is because it points out the major problem with the properties we said a good program should have. With the exception of e ciency they are very di cult to measure. Considerable research has been done in the last years to find methods to measure correctness and maintainability of software.[25] While many methods have been proposed, from a practical point of view no measures have been found that work in general, i. e., allow the comparison of unrelated code and work despite the programmers knowing about being measured. On the other hand, careful use of metrics can provide useful insights, as any anomalies in the statistics are indicators for possible problems.

Popular metrics are for example *lines of code* and the *cyclomatic complexity number* (Watson and McCabe,). Further information on software metrics can be found, for example, in Shepperd ( ), Kan ( ).

Lines of code ( ) might seem to be a very primitive way to measure anything meaningful about a program. Interestingly, none of the more elaborate metrics mentioned in the literature are significantly more successful in general. The rule of thumb is that the size of a function should not exceed about lines of code. If a function is substantially bigger than this, it may be hard to understand, di cult to test and is likely to be error prone.

The cyclomatic complexity number ( ) is the minimum number of tests that can, in (linear) combination, generate all possible paths through a function.[26] Functions with a high cyclomatic complexity have many possible paths of control flow and therefore are di cult to test. Watson and McCabe ( ) suggest limiting the cyclomatic complexity to ten and advice strongly against numbers above .

25 See for example Zuse ( ) for a survey.
26 This is not entirely correct, as iterating is not taken into account.

3.9.3 Statistics

Table . gives the total account for the source code statistics. is the total number of lines of code within functions. *Stmt.* is the total number of code statements, i. e., in C basically the number of semicolons. *Calls* is the total number of function calls. *CC* is the sum of the cyclomatic complexity numbers of all functions. *Ass.* is the total number of assert statements. *Cover* is the percentage of statements that is executed by the regression tests. We use ∅ to indicate an average.

Zimpl statistics	LOC	Stmt.	Calls	CC	Ass.	Cover
684 functions, total	11369	7520	4398	1972	1255	86%
∅ per function	16.6	11.0	6.4	2.9	1.8	
∅ statements per	0.7		1.7	3.8	6	
PerPlex statistics for comparison						
∅ per function	23.6	16.1	7.9	5.1	1.9	
∅ statements per	0.7		2.0	3.1	8	

Table . : Total account of Z source code statistics

We have also computed the corresponding numbers for . While it shares some code with Z , its purpose is completely di erent. The code basically involves an -factorization and routines to solve the resulting triangular system of equations. is currently not maintained and extended very much. Even though, the numbers are quite similar. There seems to be some evidence, also from other samples that given similar source code formatting, the ratio between and statements is nearly constant.

Tables . gives the account accumulated for each source module. The first column names the module. *#F* is the number of functions defined in the module. Next are per function the average number of lines of code, statements, function calls, cyclomatic complexity and asserts. The detailed numbers for each function can be found in Appendix B. starting on page .

From the tables we can see that *ratpresolve.c* needs more testing which might get di cult, because the functions have above average size which is also reflected in a high cyclomatic complexity number.

We should note that it is possible to find substantially di erent numbers in other codes. Routines reaching a cyclomatic complexity of in just lines of code are possible. This means the routine is practically impossible to test completely.

3.9.4 Program checking tools

Since, as we mentioned earlier, the C and C++ programming languages make it quite easy to write flawed programs, tools are available to check programs for errors. They can be mainly divided into two groups: Static source code checkers and dynamic runtime checkers.

Module	#F	∅ Lines	∅ Stmt.	∅ Calls	∅ Cycl.	∅ Ass.	Cover %
bound.c	6	7.3	3.7	2.2	2.2	1.3	100
code.c	82	8.1	4.3	2.7	1.5	0.9	85
conname.c	5	13.8	8.6	4.0	2.4	2.0	100
define.c	10	7.7	4.7	1.7	1.4	1.5	100
elem.c	18	12.6	7.7	2.7	2.2	2.0	93
entry.c	15	10.5	6.2	2.9	1.7	2.1	92
gmpmisc.c	10	15.3	10.3	3.8	2.9	0.7	80
hash.c	11	16.5	12.8	3.9	3.3	2.8	79
idxset.c	9	6.9	3.9	2.9	1.1	1.3	75
inst.c	100	23.4	16.2	12.8	3.0	1.4	93
iread.c	8	45.1	31.1	17.1	7.8	1.8	89
list.c	23	9.8	5.9	2.7	1.9	1.6	90
load.c	3	35.7	25.3	8.3	10.7	2.7	78
local.c	7	16.6	11.1	4.3	3.3	1.4	90
numbgmp.c	48	9.9	6.2	4.4	1.5	1.5	84
prog.c	7	10.4	7.0	3.9	1.9	1.6	79
rathumwrite.c	5	44.0	28.6	14.0	10.8	2.2	79
ratlpfwrite.c	3	57.7	40.7	21.0	15.7	2.7	89
ratlpstore.c	66	14.8	10.2	3.0	2.7	3.0	76
ratmpswrite.c	3	58.3	36.7	20.0	12.0	3.0	91
ratmstwrite.c	1	27.0	23.0	9.0	7.0	4.0	95
ratordwrite.c	1	34.0	28.0	11.0	9.0	4.0	89
ratpresolve.c	5	83.2	48.2	26.2	18.6	3.4	51
rdefpar.c	16	8.6	4.7	2.0	1.6	1.4	56
set4.c	30	16.2	10.4	6.8	2.8	1.7	93
setempty.c	12	8.2	4.8	2.1	1.5	1.8	76
setlist.c	18	14.2	9.3	4.4	2.9	2.7	98
setmulti.c	16	25.6	17.4	5.1	4.9	4.1	95
setprod.c	13	15.6	11.0	5.1	2.9	2.8	93
setpseudo.c	12	8.8	5.2	2.2	1.8	1.8	86
setrange.c	14	14.2	8.5	3.7	2.8	1.9	91
source.c	1	31.0	24.0	3.0	5.0	6.0	100
stmt.c	9	9.7	5.0	3.1	1.9	1.4	82
strstore.c	4	8.5	5.5	1.2	1.5	0.8	100
symbol.c	17	10.9	7.3	3.7	2.1	2.2	66
term.c	18	15.1	10.9	6.8	2.2	2.4	88
tuple.c	12	13.2	9.3	3.8	2.8	2.3	82
vinst.c	20	39.4	28.8	28.4	4.8	1.8	87
xlpglue.c	19	11.9	7.2	4.6	1.8	1.3	91
zimpl.c	7	49.6	34.6	16.7	10.4	1.6	74
∅ per function	684	16.6	11.0	6.4	2.9	1.8	

Table . : Statistics by function

Source code checkers

The first of these tools was the program (Darwin,), which verifies the source code of a program against standard libraries, checks the code for non-portable constructs, and tests the programming against some tried and true guidelines. Extended versions of are still available, for example, as part of the developer kit.

Since current C and C++ compilers can perform many of the original tasks of , enhanced versions were developed with more and deeper analyzing capabilities. [27] can check source code for security vulnerabilities and coding mistakes.

For Z we used [28], a commercially available enhanced lint which, additionally to the "normal" capabilities of lint-like programs, can also track values throughout the code to detect initialization and value misuse problems, can check user-defined semantics for function arguments and return values, and can check the flow of control for possibly uninitialized variables. It unveils all kinds of unused variables, macros, typedefs, classes, members, declarations, etc., across the entire program. Apart from finding bugs and inconsistencies, has proven to be an invaluable tool for increasing the portability of programs.

Runtime checkers

These programs mainly try to find memory access errors. They are somehow linked to the program in question and check the behavior of the code at runtime. This makes them very suitable to be part of the regression tests as a run of the test-suite can be used to check the innocuousness of the program. It is clear that to make these tests meaningful, the test-suite needs high coverage.

++[29] is a commercial tool that instruments the source code to check memory accesses. [30] is another commercial tool, which does the same by instrumenting the object code. Finally [31] is an open source tool that by using a special mode of operation of the Intel IA- processors can virtualize part of the environment the program runs in. This together with the interception of shared library calls allows - to check fully optimized programs without changing them, a feature that made the integration of into the Z regression tests easy.

Further reading

Although advertised as *Microsoft's techniques for developing bug-free C programs* Maguire ( ) lists important techniques for writing good software. Apart from good advice and interesting stories van der Linden ( ) has nice discussions on the design decisions made in C and C++. Finally, reading Libes ( ) can give illuminative insights even for experienced programmers.

---

27 http://lclint.cs.virginia.edu
28 http://www.gimpel.com
29 http://www.parasoft.com
30 http://www.ibm.com/software/rational
31 http://valgrind.kde.org

Part II

Applications

Chapter 4

Facility Location Problems in Telecommunications

In order to ask a question you must already know most of the answer.
— Robert Sheckley, *Ask a foolish question, 1953*

This chapter is a digest of the experiences from three projects: The access network planning for the German Gigabit-Wissenschaftsnetz G-WiN conducted together with the DFN (Verein zur Förderung eines Deutschen Forschungsnetzes e.V.), the mobile switching center location planning project conducted together with e·plus, and the fixed network switching center location planning project conducted together with Telekom Austria. Note that all data shown reflects the state of affairs at the time of the projects which took place between 1998 and 2002.
I wish to thank Gertraud Hoffmann and Marcus Pattloch from DFN, Erhard Winter from e·plus, Robert Totz from Telekom Austria, my colleagues Andreas Bley, Alexander Martin, Adrian Zymolka, and especially Roland Wessäly who apart from his scientific contributions helped to calm stormy waters.

In this chapter we will show some real-world examples of how to apply the modeling toolbox introduced in the previous chapters. The real-world objects we deal with are quite di erent, but we will see how they can be mapped to essentially the same mathematical model.

First a mathematical model for the hierarchical multicommodity capacitated facility location problem is introduced. Then we will present for each project how we adapted the model to its specific requirements, how we dealt with peculiarities, and note special problems that result from the decisions made in the projects. Since these are case studies we do not try to completely cover the subjects, but give illustrated "how did we do it" stories with some notes on details that deserve attention.

Please keep in mind that we are talking about real projects with industrial partners. In a perfect world (for an applied mathematician) complete and consistent data is already available at the start of a project, together with the best solution the engineers could find so far. Then for an all greenfield situation a new solution can be computed that is comparable but % (Dueck,) better.

In the real world there is no data to begin with. Then after a long time there is some questionable and inconsistent data. We prepare first solutions, discuss them with the practitioners and modify the model, the data and our attitude. Sometimes parts of solutions are fixed in between and others are reoptimized again. Comparisons with other solutions often do not make sense, because either there are no other solutions ( ), the planning goals are not the same (e·plus), or the objective is varying (T A). In this sense the following are success stories, because they succeeded even when the goals were afloat.

4.1 Traffic

Telecommunication networks are about transmitting information. There are several ways to describe the amount of transmissions or tra c present in a network. In digital networks *bits per second* (bit/s) is used to describe the data rate of a transmission. For example, to transmit a single ( ) voice call we need a connection capable of kbit/s. We call this a kbit/s *channel*.

Leased connections are usually priced according to their maximum transmission rate. Depending on the technology only certain rates might be available. Typical rates are kbit/s, Mbit/s, Mbit/s, Mbit/s, Mbit/s, and . Gbit/s.

Voice tra c is often measured in *Erlang*, which is defined as 1 Erlang := (utilization time) / (length of time interval). For example, a customer talking minutes on the phone within one hour, generates a tra c of . Erlang. The unit and therefore the amount of tra c depends on the length of the time interval. In all our projects, the time interval used was minutes. Usually those minutes during a day are chosen which bear the maximum amount of tra c. We call this the *peak hour* or *busy hour*.[1]

Assuming the calls arrive according to a Poisson process with average arrival rate λ and further assuming that the call duration process is exponentially distributed with parameter μ, the probability p_c that a call cannot be handled with c available channels can be computed as

$$p_c = \frac{\frac{\rho^c}{c!}}{\sum_{i=0}^{c} \frac{\rho^i}{i!}} \qquad \text{with } \rho = \frac{\lambda}{\mu} \ . \qquad (\ .\ )$$

We call p_c the *blocking probability*. Let us consider an example. Suppose the capacity of a link with call arrival rate of $\lambda = 720$ calls/hour and an average call duration of $1/\mu = 3$

1 Note that the total tra c in the network is used to determine the peak hour. As a result parts of the network can have higher tra c at other times. Depending on the kind of the network the time in the day and the amount of tra c might heavily fluctuate from day to day. While, for example, voice telephone networks tend to be rather stable in this regard, imagine researchers from the particle accelerator in Hamburg transmitting gigabytes of measurement data after each experiment to storage facilities of the participating institutions at other locations.

minutes/call should be determined. The resulting demand is $\rho = \lambda/\mu = 720 \cdot 3/60 = 36$ Erlang. Given channels, the blocking probability $p_{30} \approx 0.23$.

Table . lists the number of kbit/s channels needed to handle tra c demands given in Erlang depending on the required blocking probability. The highest channel per Erlang ratio can be observed for low tra c demands. The ratio changes reciprocally proportional to the blocking probability and also to the tra c demand.

Blocking probability	Traffic in Erlang 20	30	40	60	80	100	150	200	300	400	500	1,000
1%	30	42	53	75	96	117	170	221	324	426	527	1,031
5%	26	36	47	67	86	105	154	202	298	394	489	966
10%	23	32	43	62	80	97	188	242	279	370	458	909

Table . : Number of kbit/s channels depending on tra c in Erlang

For more detailed information on these topics, see Brockmeyer et al. ( ), Qiao and Qiao ( ), Wessäly ( ).

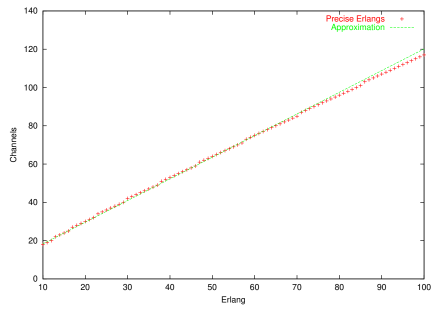
4.1.1 Erlang linearization

In our discussions with practitioners we heard a lot about Erlang curves and so-called *bundling gains*, which take place when several channels are combined. Admittedly equation . does not look the least linear. Nevertheless, for practical purposes, the function e_c, mapping Erlang to channels, can be approximated with su cient accuracy by a linear function $\tilde{e}_c$, given that the tra c in question is above ten Erlang and the blocking probability is below %.

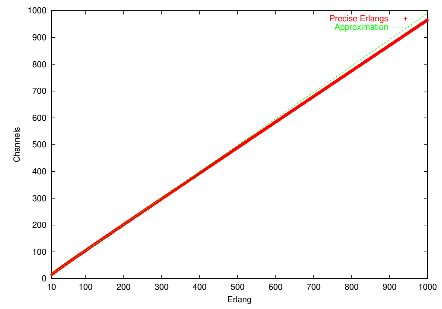
Figures . a and . b compare e_c and a linear approximation for % and % blocking probability, respectively. Figures . c and . d show the relative approximation error, i. e., $|e_c(x) - \tilde{e}_c(x)|/e_c(x)$. The interesting structure of the error figures results from e_c being a staircase function, since channels are not fractional. In the cases shown, the error of the approximation is always below %. The apparent diverging of the approximation in Figure . b results from minimizing the maximum relative error when choosing the parameters for the approximation function.

If it is possible to further limit the range of the approximation the results are even better. For example, with % blocking probability the maximum error for the interval to Erlang is about . %, and for the interval to , Erlang it is only . %.

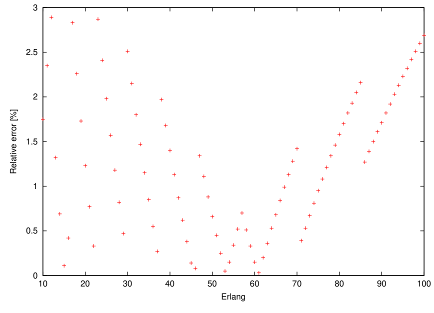
We conclude with the observation that for practical relevant blocking probabilities ($\leqslant$ %) and for tra c demands ($>$ Erlang) as they occur in the backbones of voice networks, a linear approximation of the numbers of channels needed is acceptable. Further, it can be concluded from the point of intersection of the approximated function with the y-axis that bundling two groups of channels saves approximately about ten channels. (Note that the origin is not drawn in the figures.)



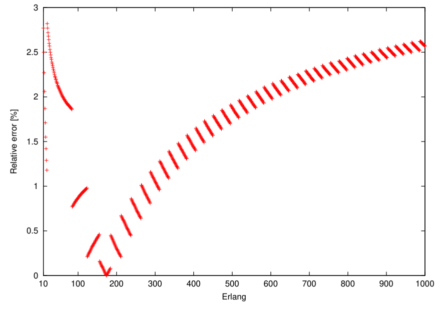
(a) Linear approximation $\tilde{e}_c = 1.1315\rho + 7$, for $\rho = 10, \ldots, 100$ Erlang with $p_c = 0.01$



(b) Linear approximation $\tilde{e}_c = 0.98525\rho + 5.5625$, for $\rho = 10, \ldots, 1000$ Erlang with $p_c = 0.05$



(c) Relative error of linear approximation



(d) Relative error of linear approximation

Figure . : Linear approximation of the Erlang to channel function

4.1.2 Switching network vs. transport network

The switching network consists of the logical links between nodes in a network, while the transport network consists of the physical links. Switching networks as we describe them in this chapter are hierarchical, essentially tree-like, networks. Transport networks in contrast are usually meshed networks.

The interrelation between the amount of tra c in the switching network and the amount of tra c in the corresponding transport network is a complex one. Nearly all aspects of the network, like routing, protocols, and connection types, are di erent between the switching and the transport network. Details can be found, for example, in Wessäly ( ). It is the transport network where real costs for physical installations arise. Since the ties to the switching network are so vague, it is very di cult to associate meaningful costs with links in the switching network. We will discuss this in greater detail specifically for the projects.

4.2 A linear mixed integer model for hierarchical multi-commodity capacitated facility location problems

Given a layered directed graph $G = (V, A)$ with N hierarchy-levels $L = \{1, \ldots, N\}$. The nodes are partitioned into layers as $V_1, \ldots, V_N$, with $V_m \cap V_n = \emptyset$ for all $m, n \in L$, $m \neq n$, and $V = \bigcup_{n \in L} V_n$. Without loss of generality we will assume $|V_N| = 1$ and denote $r \in V_N$ as the *root*.[2] The nodes are connected with arcs $A \subseteq \{(u, v) \mid u \in V_n, v \in V_{n+1}, \text{ and } n, n+1 \in L\}$. Figure . shows an example with $N = 4$.

We are looking for a tree that connects all level one nodes with the root. Since G is layered this means that each level one node has to be connected to exactly one level two node. These in turn have to be connected to exactly one level three node and so on. This is essentially the problem of finding a Steiner tree in a graph (see also chapter) with root r and V_1 as terminal set. The red arcs in Figure . mark a possible solution. All nodes from level one and level N are always part of the solution.

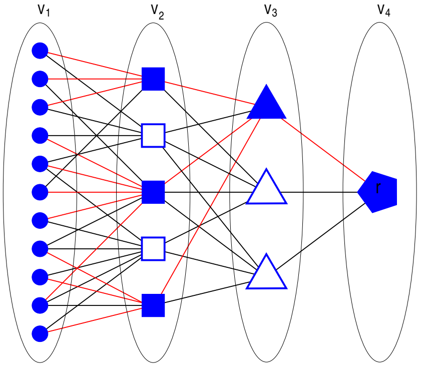


Figure . : Layered graph with $N = 4$

For each node $v \in V$ and each arc $(u, v) \in A$ we introduce a binary variable y_v and x_{uv}, respectively. Each y_v and each x_{uv} is equal to one if and only if the node or arc is active, i. e., is part of the solution. This leads to the following formulation:

$$y_v = 1 \qquad \text{for all } v \in V_1 \qquad (.)$$

$$x_{uv} \leqslant y_v \qquad \text{for all } (u, v) \in A \qquad (.)$$

$$\sum_{(v,w) \in A} x_{vw} = y_v \qquad \text{for all } v \in V \setminus \{r\} \qquad (.)$$

Note that (.) implies $x_{vw} \leqslant y_v$ and that for $r \in V_N$ the above system implies $y_r = 1$.

2 This can always be achieved by introducing a single node in an additional layer which is connected to all nodes of the previous layer.

Commodities

For each node $v \in V$ and each $d \in D$ of a set of commodities (resources), a demand $\delta_v^d \geqslant 0$ is given, specifying the demand which has to be routed from each node $v \in V$ to the root node $r \in V_N$. This can be modeled by introducing a non-negative continuous variable f_{uv}^d, $(u, v) \in A$ denoting the amount of flow of commodity $d \in D$ from node u to v:

$$\delta_v^d + \beta_v^d \sum_{(u,v)\in A} f_{uv}^d = \sum_{(v,w)\in A} f_{vw}^d \quad \text{for all } v \in V \setminus \{r\}, d \in D \qquad (\ .\ )$$

$\beta_v^d > 0$ is a "compression" factor, i. e., all incoming flow into node v of commodity d can be compressed (or enlarged) by β_v^d. We will see some applications for this factor later on. If we assume all β_v^d to be equal within each layer, the total amount of flow of each commodity reaching the root will be constant. Note that for any $v \in V_1$ equation (.) reduces to $\delta_v^d = \sum_{(v,w)\in A} f_{vw}^d$.

For each arc $(u, v) \in A$ the flow of commodity $d \in D$ given by f_{uv}^d has an upper bound $\rho_{uv}^d \geqslant 0$. Since flow is allowed only on active arcs, i. e.,

$$\rho_{uv}^d x_{uv} \geqslant f_{uv}^d \quad \text{for all } (u, v) \in A, d \in D \ , \qquad (\ .\ )$$

the upper bound does not only limit the capacity of the arcs, but due to (.) also the capacity of node u, since the flow going into u has to leave on a single arc. Note that for all nodes $v \in V_1$ with $\delta_v^d > 0$ for any $d \in D$ equation (.) is redundant due to (.) and (.).

Configurations

For each node $v \in V$ a set of configurations S_v is defined. Associated with each configuration $s \in S_v$ is a capacity κ_s^d for each commodity $d \in D$. We introduce binary variables z_{vs} for each $s \in S_v$ and each $v \in V$. A variable z_{vs} is one if and only if configuration s is selected for node v. For each active node a configuration with su cient capacity to handle the incoming flow is required:

$$\sum_{s\in S_v} z_{vs} = y_v \qquad \text{for all } v \in V \qquad (\ .\ )$$

$$\delta_v^d + \beta_v^d \sum_{(u,v)\in A} f_{uv}^d \leqslant \sum_{s\in S_v} \kappa_s^d z_{vs} \qquad \text{for all } v \in V, d \in D \qquad (\ .\ )$$

Of course, for all level one nodes the configuration can be fixed in advance, since there is no incoming flow apart from δ_v^d.

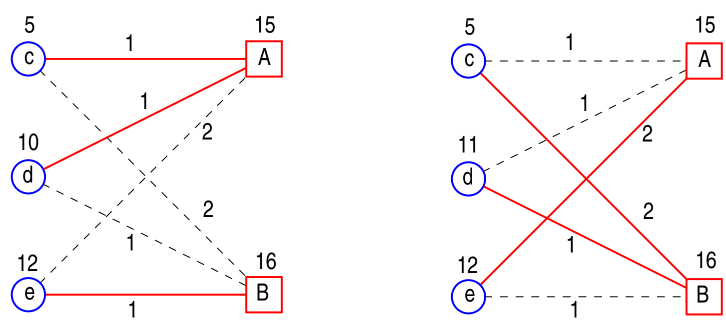
Sometimes only links with discrete capacities out of a set K_d of potential capacities regarding a certain commodity $d \in D$ are possible. This can be modeled by introducing binary variables $\bar{x}_{uv}^{dk}$ for each commodity $d \in D$, each link capacity $k \in K_d$ and for each link $(u,v) \in A$ and adding the following constraints:

$$\sum_{k \in K_d} \bar{x}_{uv}^{dk} = x_{uv} \qquad \text{for all } (u,v) \in A, d \in D \qquad (\ . \ )$$

$$\sum_{k \in K_d} k\,\bar{x}_{uv}^{dk} \geqslant f_{uv}^{d} \qquad \text{for all } (u,v) \in A, d \in D \qquad (\ . \ )$$

Equation (.) ensures that for each active arc exactly one of the possible capacities is chosen. Inequality (.) makes sure that the link has su cient capacity. Note that depending on the particular problem simplifications are possible, especially regarding (.) and (.), and (.), (.) and (.).

Configurations, as all types of (hard) capacity constraints, can lead to instable solutions, i. e., solutions that vary considerably upon small changes of the input data. Figure . shows an example. Given are three nodes c, d, and e with demands $\delta_v = 5, 10$, 12, respectively. The two serving nodes A and B have a fixed capacity of 15 and 16, respectively. The costs for connecting the demand nodes with the serving nodes is drawn along the connections. Figure . a shows the optimal solution when minimizing connection costs. Now, an increase in the demand of node d by one results in the solution shown in Figure . b, that is, changing a single demand by a small amount leads to a completely di erent solution. This is highly undesirable, since, as we will see in the next sections, the input data is usually inaccurate.



(a) Optimal solution (b) After a small change

Figure . : Instable solution

Apart from being unstable, solutions where nodes are not connected to the cheapest available higher level node just look wrong to the practitioners. To prevent this, inequalities like

$$x_{uv} \leqslant 1 - y_w \quad \forall (u,v) \in A, w \in V \text{ with } c_{uv} > c_{uw}$$

can be introduced, where c_{uv} for $(u,v) \in A$ denotes the costs associated with a connection between node u and node v.

Objective function

The objective is to minimize the total cost of the solution, i. e.,

$$\min \sum_{v \in V} \big(y_v + \sum_{s \in S_v} z_s\big) + \sum_{(u,v) \in A} \Big(x_{uv} + \sum_{d \in D} \big(f^d_{uv} + \sum_{k \in K_d} \bar{x}^{dk}_{uv}\big)\Big)$$

with appropriate objective function coe cients for all variables.

Literature

The capacitated facility location problem is well studied and of considerable importance in practice. As we mentioned before, it can also be seen as a capacitated Steiner arborescense problem, or as a partitioning or clustering problem. Many variations are possible. As a result a vast amount of literature on the problem, variations, subproblems, and relaxations has been published. See, for example, Balakrishnan et al. ( ), Hall ( ), Mirchandani ( ), Bienstock and Günlück ( ), Aardal et al. ( ), Ferreira et al. (,), Park et al. ( ), Holmberg and Yuan ( ), Ortega and Wolsey ( ), Gamvros and Golden ( ), Bley ( ). It should be noted that the majority of the publications is not related to real-world projects.

4.3 Planning the access network for the G-WiN

In we got involved into the planning of what should become Germany's largest network, the Gigabit Research Network G-WiN operated by the . All major universities and research facilities were to be connected. The network was planned to handle up to tra c per hour in its first year. An annual increase rate of . was anticipated, leading to a planned capacity of about , in .

Since the is not itself a carrier, i. e., does not own any fiber channels, a call for bids had to be issued to find the cheapest carrier for the network. European law requires that any call for bids exactly specifies what the participants are bidding on. This means the had to come up with a network design before calling for bids.

As a result it was decided to design some kind of sensible network and hope the participants of the bid were able to implement it cheaply. The network should consist of backbone nodes. Ten of these backbone nodes should become interconnected *core* nodes, while the other backbone nodes should be connected pairwise to a core node. We will see later on in Section . that the decision to have ten core nodes was probably the most important one in the whole process. For information on the design of the network connecting the core nodes see Bley and Koch ( ), Bley et al. ( ).

In this case no distinction between transport network and switching network was necessary, as the bid was for the logical or virtual network as specified by the . The mapping of logical to physical connections was left to the provider of the link. As a result no pricing information for installing links between the nodes was available before the bid. It was decided to use costs according to those of the predecessor network B-WiN, but scale them by some factor to anticipate declining prices. Bee-line distances between the locations were used as link distances. Since the hardware to be installed at the nodes was either unknown, not yet available from the vendors, or depending on the carrier, no real costs or capacities were known.

The initial problem for the access network was given as follows: *Having 337 nodes from which 224 are potential backbone nodes, select 30 backbone nodes and connect each of the remaining nodes to them.*

Note that selecting the ten core nodes was not part of the problem. Connections to backbone nodes had to have one of the following discrete capacities: kbit/s, Mbit/s, Mbit/s, Mbit/s, . Gbit/s, or Gbit/s. Initially clients demanding kbit/s were not considered and none of the clients needed more than Mbit/s. The associated cost function is shown in Figure . . Additionally Figure . visualizes for each location the demand for the peak tra c hour.

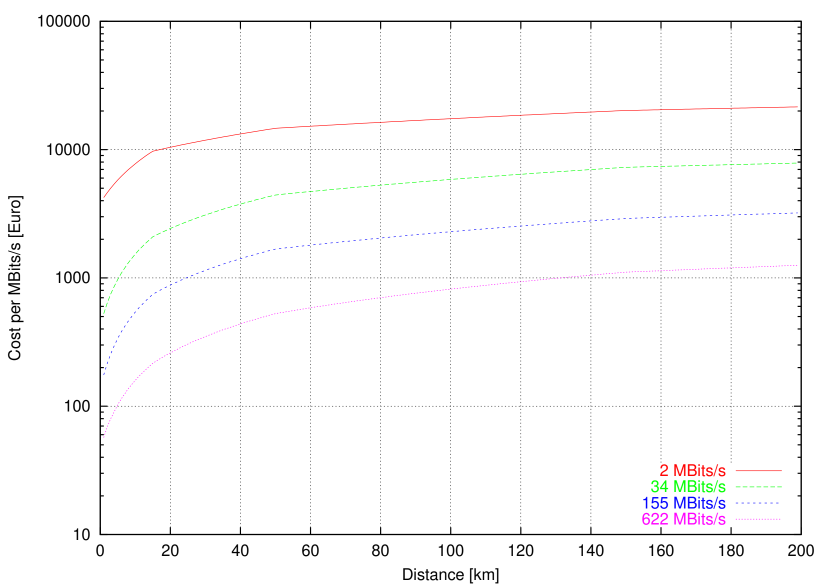


Figure . : Cost depending on the distance per Mbit/s

Since the hardware installed at the backbone nodes has to operate hours, seven days a week, places with suitable maintenance, air conditioning and uninterruptible power supplies are required. While of the sites were capable in principle to host the hardware, some were preferred. We modeled this by decreasing the cost for the preferred nodes by a small margin.

But even for the preferred locations, the conditions for hosting the equipment had to be negotiated. This led to iterated solutions with consecutively more and more fixed sites. In the end the problem degenerated to a pure assignment problem. For the same reasons the selection of the core nodes was done at . Finally we also added the kbit/s clients to the problem, bringing the total number of locations to .

We said in the beginning that the annual increase in tra c was to be taken into ac-

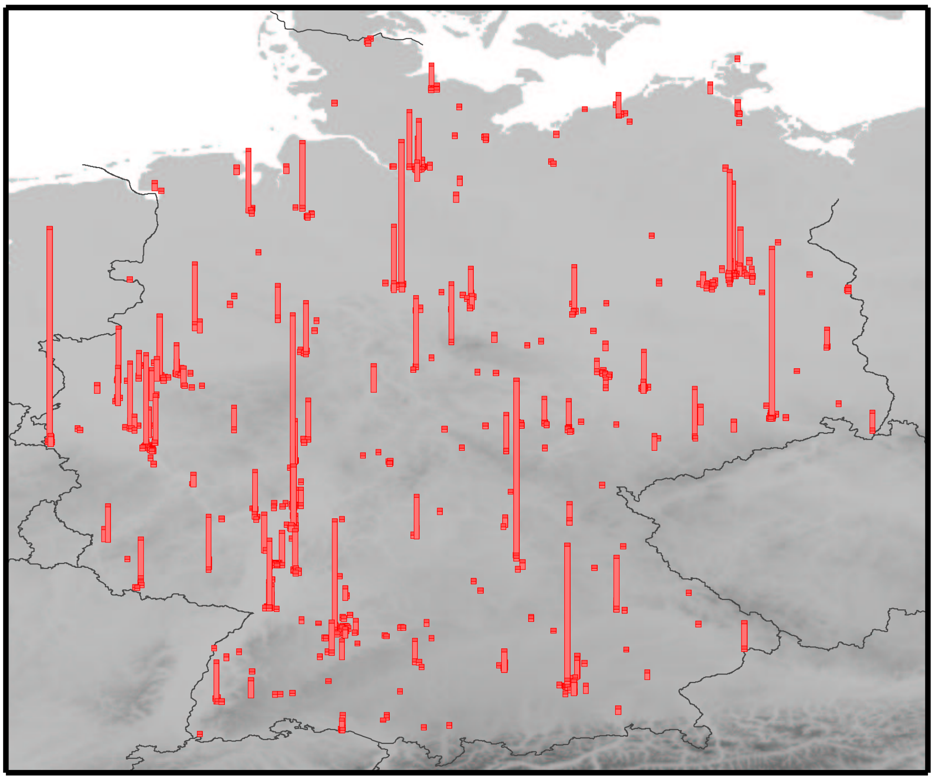


Figure . : Demands of G-WiN locations

count. Since the increase was given as a linear factor on all tra c, the only change could be due to the discretization of the link capacities. But it turned out that the resulting di erences were negligible.

Modeling

We modeled the problem with three layers and discrete link capacities between the backbone and the core nodes. Using this model, we compare our original solution to a less restricted one, where the optimization can decide where to place the core nodes.

V_1 is the set of all demand nodes. Potential backbone nodes are split into a client part, carrying the demand and belonging to V_1 and a backbone part belonging to V_2. The set of potential core nodes is denoted V_3. While the three sets are disjunctive, their elements might refer to the same physical locations. The function $\sigma(v), v \in V_1 \cup V_2 \cup V_3$ maps nodes to their corresponding location.

The set of arcs is defined as $A \subseteq (V_1 \times V_2) \cup (V_2 \times V_3)$. The variables are defined similarly to Section . ; in particular $x_{uv}, (u, v) \in A \cap (V_1 \times V_2)$ are binary variables denoting which connections are active, $\bar{x}^k_{vw}, (v, w) \in A \cap (V_2 \times V_3), k \in K$ are binary variables denoting which capacity is used for a link between a backbone and a core node.

In addition to the binary variables $y_v, v \in V_2$, denoting the active backbone nodes, a second set of binary variables $\bar{y}_w, w \in V_3$, denoting the active core nodes is introduced. Since only a single commodity is present, the commodity index $d \in D$ is dropped. The following model describes the problem setting:

$$\sum_{(u,v)\in A} x_{uv} = 1 \qquad \text{for all } u \in V_1 \qquad (\;.\;)$$

$$x_{uv} \leqslant y_v \qquad \text{for all } u \in V_1, (u,v) \in A \qquad (\;.\;)$$

$$\sum_{(v,w)\in A} \sum_{k\in K} \bar{x}^k_{vw} = y_v \qquad \text{for all } v \in V_2 \qquad (\;.\;)$$

$$\sum_{k\in K} \bar{x}^k_{vw} \leqslant \bar{y}_w \qquad \text{for all } v \in V_2, (v,w) \in A \qquad (\;.\;)$$

$$\sum_{(v,w)\in A} \sum_{k\in K} k\bar{x}^k_{vw} \geqslant \sum_{(u,v)\in A} \delta_u x_{uv} \qquad \text{for all } v \in V_2 \qquad (\;.\;)$$

$$\bar{y}_w \leqslant y_v \qquad \text{for all } v \in V_2, w \in V_3, \sigma(v) = \sigma(w) \qquad (\;.\;)$$

$$\sum_{w\in V_3} \bar{y}_w = 10 \qquad (\;.\;)$$

$$\sum_{(v,w)\in A} \sum_{k\in K} \bar{x}^k_{vw} = 3\bar{y}_w \qquad \text{for all } w \in V_3 \qquad (\;.\;)$$

Note that (.) results from combining (.) with (.). Inequality (.) corresponds to (.), while (.) is a combination of (.) and (.), and (.) is a combination of (.) and (.). Inequality (.) is a simplified form of (.). Inequality (.) ensures that only chosen backbone nodes can become core nodes. The number of core nodes is fixed to ten by equation (.). Finally, equation (.) fixes the number of backbone nodes to , with the additional requirement that one of every three backbone nodes has to become a core node with two other backbone nodes attached to it. The objective function is

$$\min \sum_{v\in V_2} c_v y_v + \sum_{w\in V_3} c_w \bar{y}_w + \sum_{\substack{(u,v)\in A\\ u\in V_1}} c_{uv} x_{uv} + \sum_{\substack{(v,w)\in A\\ v\in V_2}} \sum_{k\in K} c^k_{vw} \bar{x}^k_{vw}$$

with c_v, $v \in V_2$ denoting the cost for selecting v as backbone node and c_w, $w \in V_3$ denoting the cost for selecting node w as core node. The cost for connecting demand node $u \in V_1$ to backbone node $v \in V_2$ is given as c_{uv} and the cost for connecting backbone node $v \in V_2$ to core node $w \in V_3$ with capacity $k \in K$ is denoted by c^k_{vw}.

The corresponding Z program can be found in Appendix C. on page . We call this the *normal* scenario. We also examined a *relaxed* scenario, where equations (.) and (.) are removed. Referring to the *original* scenario means the solution used in the project.

Results

nodes were considered. nodes were potential backbone nodes, including preferred ones, giving them a small bonus in the objective function. Only connections between demand nodes and backbone nodes of less than kilometers were allowed. Backbone nodes that were attached to core nodes had to be at least kilometer apart from the core node. The cost for opening a backbone node was set to . Opening a core node again involved a cost of , or for a preferred node. The resulting integer program for the normal scenario has , binary variables, , constraints and , non-zero entries in the constraint matrix.

Scenario	Gap [%]	Time [h]	BB	Core	Objective
Normal	7.12	18	30	10	67,593
Relaxed	0.28	3	16	15	58,022
Original	— [3]	— [3]	29	10	67,741

Table . : G-WiN solution

Table . lists the results for the di erent scenarios. *Gap* shows the optimality gap of the solution. *Time* is the approximate time spent by . for solving the instance. *BB* and *Core* give the number of backbone and core nodes, respectively. *Objective* lists the objective function value for the scenarios. The cost for the *original* scenario is almost equal to the cost for the *normal* scenario, indicating that given the number of backbone and core nodes is fixed in advance, the original solution is less than % o the optimum.

Figure . shows images of the results. Backbone nodes are marked as green circles, core nodes are drawn as red triangles. The picture indicates that the cost for the backbone to core node links was set too high to make them pay o . While the relaxed scenario seems to incur the least cost, keep in mind that we have not included any costs for the core network and that the objective value for the relaxed scenario is only % smaller than for the original one.

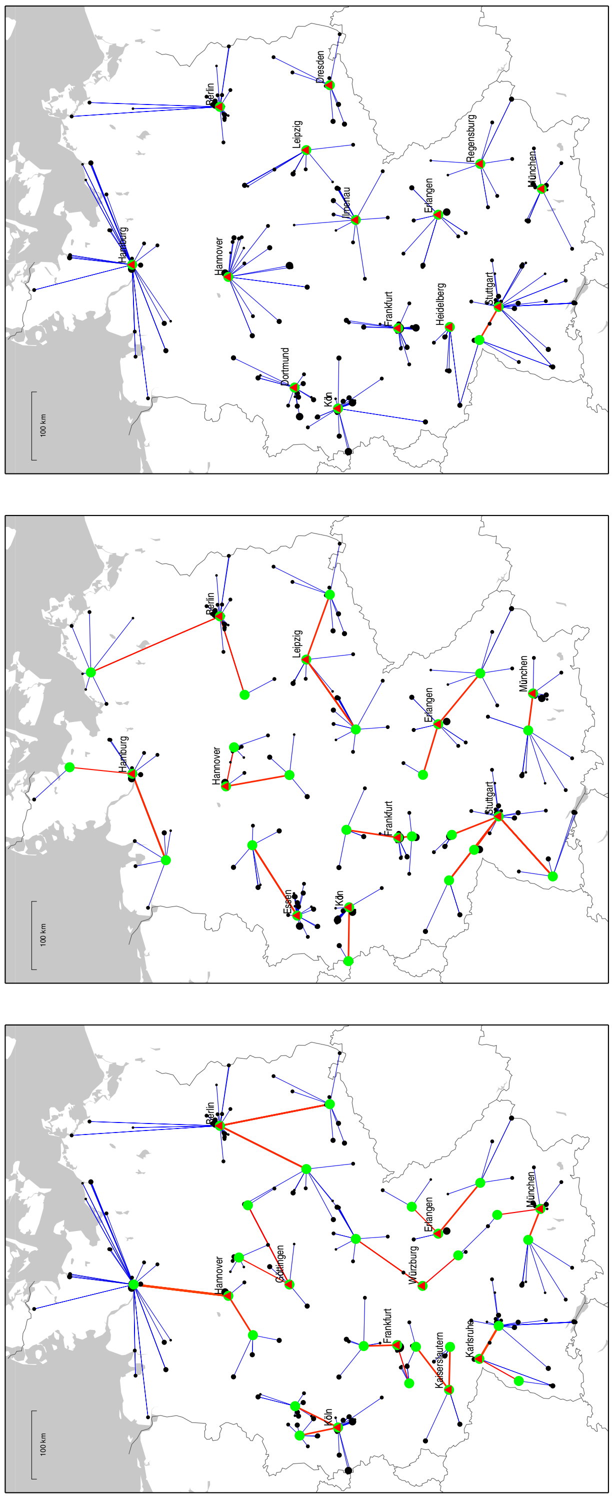
Epilogue

The bid for the carrier was won by the German Telekom.

At least one of the persons involved in the design and planning of the network had to retire with a mental breakdown.

By now the G-WiN is running very successfully for more than four years and has been reconfigured and upgraded several times. Between and we investigated the profitability of introducing a third layer of backbone nodes and discovered some potential candidates.

3 No gap and time are given for the original scenario, because it was interactively refined by selecting backbone nodes until all decisions were taken. The selection of the core nodes was done by the .



(a) Normal

(b) Original

(c) Relaxed

Figure . : Results for the G-WiN access network planning.

4.4 Planning mobile switching center locations

This project, which we conducted together with e·plus, examined the logical layout of a part of a network. The signal from a mobile is transmitted to an antenna that is located at a *Base Transceiver Station* ( ). The is connected to a *Base Station Controller* ( ). The manages the transmitter and receiver resources for the connected base stations and controls the tra c between the base station and the *Mobile Switching Center* ( ). The s are at the core of the network and interconnect all the s via connections to the other s in the network. s are essentially computers that can be built with di erent capacities. One resource limiting the capacity of a is the number of *subscribers*.[4] Each , depending on the tra c it manages, takes up a number of subscribers. The installation cost of a depends on its subscriber capacity. The connection costs between a and an depend on the data rate of the link. Since e·plus owned only part of its transport network and leased links on demand, it was di cult to associate costs to links in a combinatorial way. The price for each new link had to be individually investigated. As a result we tried di erent cost functions within the project, either similar in appearance to the one given in Figure . , or just a linear function depending on the capacity and the distance.

We can state the problem as follows: *Given a list of s, a list of potential locations, and a list of possible configurations, decide where to place s and for each to which it should be connected. Choose a suitable configuration for each* .

Model

The problem can be formulated using a simplification of the model given in Section . :

$$\begin{aligned} \sum_{(v,w)\in A} x_{vw} &= 1 && \text{for all } v \in V_1 \\ \sum_{s\in S_v} z_{vs} &= 1 && \text{for all } v \in V_2 \qquad (\, . \,) \\ \sum_{(u,v)\in A} \delta_u x_{uv} &\leqslant \sum_{s\in S_v} \kappa_s z_{vs} && \text{for all } v \in V_2 \end{aligned}$$

V_1 is the set of s and V_2 is the set of potential s. δ_u denotes for each $u \in V_1$ the number of associated subscribers. For each $v \in V_2$ the parameter $\kappa_s, s \in S_v$ denotes the number of subscribers which can be served by configuration s.

Note that (.) requires a "zero" configuration, i. e., there has to be exactly one $s \in S_v$ with $\kappa_s = 0$ for each $v \in V_2$. This has the nice e ect that already existing configurations can be modeled this way. Instead of assigning the "building cost" to a configuration, the cost involved with a particular change is used.

4 Technically, this is not entirely correct. Attached to each is a *Visitor Location Register* ( ), a database which actually imposes the restriction on the number of subscribers. For our purposes it su ces to view the as part of the .

For the computational study described in the next section the following objective function was used:

$$\min \sum_{(u,v)\in A} \mu\, d_{uv}\, l_{uv}\, x_{uv} + \sum_{v\in V_2} \sum_{s\in S_v} c_{vs} z_{vs}$$

with μ being a predefined scaling factor, d_{uv} denoting the bee-line distance between $u \in V_1$ and $v \in V_2$ in kilometers, and l_{uv} denoting the number of kbit/s channels needed for the tra c between u and v. The building cost for configuration $s \in S_v$ at node $v \in V_2$ is denoted by c_{vs}.

Results

We computed solutions for ten di erent scenarios. Table . lists the parameters that are equal in all cases. The scenarios are partitioned into two groups. The number of subscribers in the first group was . million, and . million in the second group. For each group five solutions with di erent connection costs were computed, using $\mu =$, , , , . The Z program used can be found in Appendix C. on page . All runs were conducted with . using default settings[5].

Number of BSCs $\lvert V_1\rvert$	212
Number of potential MSCs $\lvert V_2\rvert$	85
Maximum allowed BSC/MSC distance [km]	300
Minimum MSC capacity (subscribers)	50,000
Maximum MSC capacity (subscribers)	2,000,000
Number of different configurations $\lvert S_v\rvert$	14
Binary variables	10,255
Constraints	382
Non-zero entries in constraint matrix	20,425
CPLEX time limit	1 h

Table . : Scenario parameters

Table . and Figure . show the result for the scenarios. *Gap* lists the gap between the primal solution and the best dual bound when either the time limit of one hour was reached, or ran out of memory. *MSC* is the number of s that serve s. ∅ *util.* is the geometric mean of the utilization of the s. *Hardw. cost* is the sum of the cost of the configurations ($= \sum_{v\in V_2} \sum_{s\in S_v} c^z_s z_s$). *Chan.×km* is the sum of the number of kbit/s channels times the distance in kilometers needed to connect all

5 In one case it was necessary to lower the *integrality tolerance* from 10^{-5} to 10^{-8}. This is a scaling problem. It probably could have been avoided if we had used multiples of thousand subscribers for the demands and capacities.
It seemed that the default automatic setting for cut generation sometimes added more and often better cuts than setting explicitly to aggressive cut generation. For the . million user scenario with $\mu = 1$, the default setting generated cover cuts, clique cuts and cover cuts increasing the objective from initially $2.3866 \cdot 10^7$ to $3.2148 \cdot 10^7$. Branching , nodes only increased the lower bound further to $3.2655 \cdot 10^7$.

s ($= \sum_{(u,v)\in A} d_{uv} l_{uv} x_{uv}$). *Total cost* is the objective function value. All cost figures given are divided by , and rounded to improve clarity.

The results are not surprising. The higher the connection costs the more s are opened. Most of the results show the problem we mentioned above, that s got connected to remote s to circumvent upgrading the nearer ones.[6]

In an earlier similar study, (Wessäly,) was used in a subsequent step to design and dimension the inter- transport network for each of the solutions. As it turned out, the costs for the inter- network varied highly between the scenarios and dominated the costs for the - network by a huge amount.

From this result we concluded that some interaction between the location planning and the planning of the inter- backbone network is necessary. A possible solution might be to assign costs to the connections between the V_2 nodes (s) and a virtual root. But since this corresponds to a star shaped backbone network it is not clear if it is possible to find suitable costs that resemble a real backbone network somehow. An integrated model, as presented in Bley et al. ( ), seems to be more promising here.

μ	Fig.	Gap [%]	MSC	∅ util. [%]	Hardw. cost	Chan. × km	Total cost
3.4 million subscribers							
1	4.7a	4.66	4	99.4	23,850	1,612	25,462
5	4.7b	3.74	7	99.0	25,884	772	29,744
10	4.7c	1.96	8	98.7	26,716	659	33,309
20	4.7d	0.00	12	98.6	29,335	486	39,059
100	4.7e	0.00	32	93.9	43,199	191	62,265
6.8 million subscribers							
1	4.7f	2.78	5	99.8	46,202	1,987	48,189
5	4.7g	1.49	8	99.7	47,952	1,179	53,846
10	4.7h	0.30	11	99.6	49,636	926	58,897
20	4.7i	1.12	19	97.5	55,473	570	66,873
100	4.7j	0.13	40	96.7	72,200	250	97,199

Table . : Results of the location planning

Epilogue

The model was implemented for e·plus using Microsoft as a front- and backend, in a chain →Z → →awk → .

The implementation allowed also three tier models, e. g. - - , or - - Data-center. Unfortunately at the time of installation, no data was available for these models.

6 This can also happen if the instances are not solved to optimality. It is therefore necessary to post-process solutions before presenting them to practitioners, making sure the solutions are at least two-optimal regarding connection changes. Presenting visibly suboptimal solutions can have embarrassing results, even when the links in question have a negligible influence on the total objective.

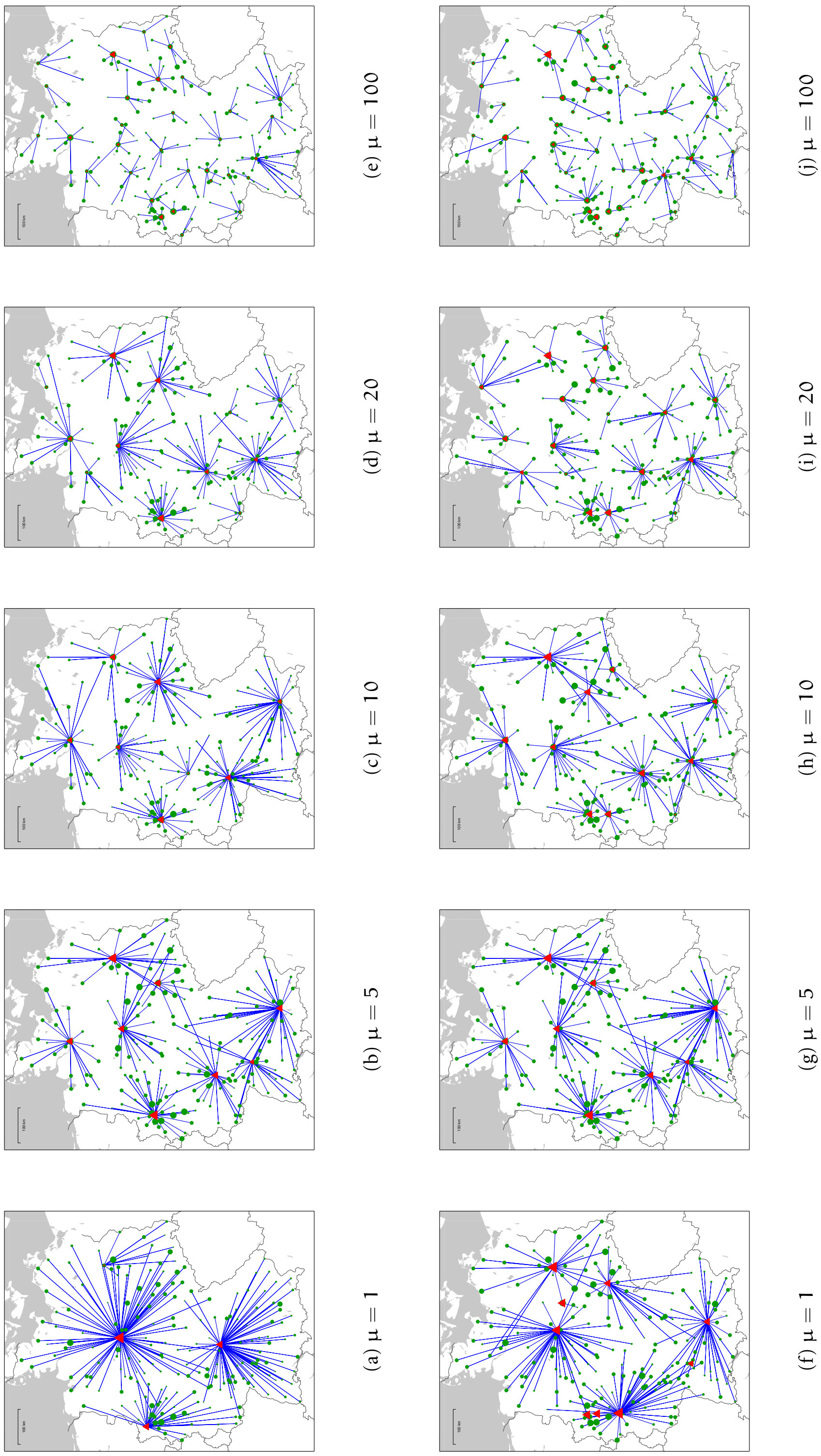


Figure . : Results for the location planning. Upper row . million subscribers, lower row . million subscribers

4.5 Planning fixed network switching center locations

Telephone networks are so-called *circuit switched* networks, i. e., if one terminal is calling another terminal, the request is transmitted first to the appropriate switching center, which, depending on the location of the destination terminal, selects (switches) the route to the next switching center. This is repeated until the destination terminal is reached. In a circuit switched network this route between the two terminals is created at the beginning of the transmission and stays alive until the call is finished. The required bandwidth remains reserved all of the time.

The switching network of the T A has a hierarchical design. Seven *Main Switching Centers*[7] ( ) are the backbone of the network. On the level below are about *Network Switching Centers*[8] ( ). Next are about *City Switching Centers*[9] ( ) and finally on the bottom level are about , *Passive Switching Centers*[10] ( ).

The topology of the network is basically a tree apart from the s which are linked directly to each other. All other switching centers have to be connected to a center on a higher level than themselves.

s and s are so-called *Full Switching Centers*[11] ( ) because they are able to handle internal tra c themselves, i. e., tra c that does not need to be routed higher up in the hierarchy. In contrast to this s transfer all tra c to their respective .

One of the first decisions in the project was to reduce the hierarchy of the switching network to three levels. Therefore, we drop the distinction between s and s and speak only of s. Have a look at Figure . . The red circle is an . The green squares are s and the black triangles mark s. The blue lines are the logical connections between switching centers, while the gray lines show the physical transport network.

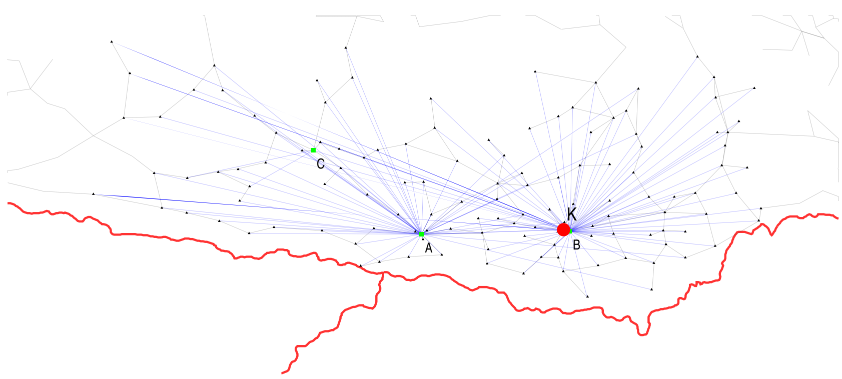


Figure . : Switching network

7 Hauptvermittlungsstellen
8 Netzvermittlungsstellen
9 Ortsvermittlungsstellen
10 Unselbstständige Vermittlungsstellen
11 Vollvermittlungsstellen. Technically s are also full switching centers, but we will use the term only for s and s.

Due to technical advances the capacity of a single switching center has been vastly increased in the last years. At the same time the cost for the transport network has steadily declined. Since the operating costs for maintaining a location are high, a smaller number of switching centers is desirable.

The goal of the project was: *To develop planning scenarios for the reduction of the number of full switching centers. For each switching center it is to decide, whether it should be up- or downgraded and to which other center it should be connected.*

Since this is not a green-field scenario, changing a switching center either way induces some cost. It is to note that the switching centers are built by two di erent manufacturers, i. e., they are not compatible below level. Only switching centers of the same manufacturer can be connected to each other.

4.5.1 Demands and capacities

The capacities of the switching centers are limited by the number of users connected and by the amount of tra c to be switched.[12]

There are three possible terminals connected to an : (Plain Old Telephone Service), -basic-rate (Integrated Services Digital Network), and -primary-rate. Only and -basic-rate draw from the restriction on the number of users of the switching centers. Assigned to each terminal type is a typical amount of tra c in Erlang. This is converted along the Erlang formula (see page) to kbit/s (voice) channels. All tra c is assumed to be symmetric between the terminals.

As noted before, all non-passive switching centers can route internal tra c directly. This can reduce the amount of tra c to the s and within the backbone. Finding an optimal partitioning of the network to minimize the external tra c is $\mathcal{NP}$-hard[13] and requires a complete end-to-end tra c matrix. Since end-to-end tra c demands were not available in the project, it was not possible to precisely model this e ect. But there is quite accurate empirical knowledge on the percentage of external tra c for each region. So it is possible to attach a fixed tra c reduction factor β (see Section) to each to approximate the e ect.

Another method to reduce the tra c in the higher levels of the hierarchy are *direct-bundle*[14] connections. These are short-cut connections from one full switching center to another which only route the direct tra c between these two switching centers.

The question whether the installation of a direct-bundle connection justifies the cost is di cult to decide. It is to be expected that the influence of the decision on the transport network is rather small, since the data is likely to use a similar route in the

12 There was another restriction called *Zoning Origins* (Verzonende Ursprünge) of which each full switching center had only a limited number. Since the whole subject was rather archaic and the numbers were somewhat hard to compute precisely, it was decided later in the project to drop the restriction, even though it would have fitted easily into the model.
13 Depending on how the problem is formulated, this is a variation of the partitioning or multi-cut problem. Further information can be found for example in Garey and Johnson ( ), Grötschel and Wakabayashi ( ), Chopra and Rao ( )
14 Direktbündel

transport network in either case. A direct-bundle needs more channels, e. g. to transmit Erlang with % blocking probability channels are needed, while in the route through the hierarchy, where much more tra c is accumulated, for example, only channels are needed to transmit Erlang. On the other hand, direct-bundles may reduce the amount of hardware needed at the level.

Given that linear costs are available for all links, it is easy to see for a single direct-bundle connection whether it pays o or not. Since the hierarchical switching network is needed anyhow, the cost for the direct-bundle can be computed and compared to the savings in the hierarchical network. The only uncertainty comes from the question whether the installation of several direct-bundle connections saves hardware cost in the hierarchy. We did not pursue the matter of direct-bundle connections any further in the project, because the question can be answered on demand for a single connection and without a complete end-to-end tra c matrix only guesses about the amount of tra c between two switching centers are possible.

To give an idea about the scenario, here are some numbers: More than three million users are distributed about : : onto , -basic-rate, and -primary-rate terminals. The tra c demand per terminal is about . , . , and . Erlang, respectively. The total tra c is more than , Erlang. A can serve about , users and , channels. The capacity of an is about ten times as big. These are average numbers as switching centers can be configured in various ways.

4.5.2 Costs

Hardware costs for installing, changing, and operating switching centers are relatively easy to determine. The biggest problems are:

- How to assess hardware that is already deployed and paid for?
- How to assess the depreciation of the purchase cost, if it is to be included at all?
- If di erent configurations for switching centers are possible, a price tag can be assigned usually only to a complete setup.

From the above it becomes clear that there is no such thing as a "real cost" in a non green-field scenario (and maybe not even then). But we can at least try to use prices that fit the goal of our investigation.

As we mentioned in Section . . we have to distinguish between the transport network and the switching network. The switching network is a logical network that creates the circuits between the terminals by building paths of logical connections between switching centers. The transport network is the physical network below that transmits the data. Now computing costs based on the tra c is di cult, because the transport network already exists in this case and the relation between tra c demands and changes in the switching network is not clear.[15] Assuming that the transport network is able to

15 Given end-to-end tra c demands and a tool like it would be possible to study the capacity requirements of the transport networks for selected scenarios.

cope with the tra c demand as induced by the current switching network and assuming further that our "optimized" switching network does not require an extension of the transport network, no real costs will occur. It follows, that the cost optimal switching network has the minimum number of switching centers possible according to the capacity restrictions. The question where a switching center should be connected to could be mostly neglected.

Nevertheless the transport network has a cost associated to it. Assuming that the amount of voice calls is rather static, any excess capacity can be used for other services, e. g., packet data services. As a result some cost function is needed, but can be arbitrarily defined.

After some discussions T A supplied the cost function shown in Figure . . The basic idea is to pay a base price per channel for the existing fiber optic cables. Since these cables need repeaters every km, which induce operating costs, the price is raised after and km. The reason for the higher price of the to connections results from the higher infrastructure demands of these links due to the higher capacities needed.[16] Note that since we usually assume about % internal tra c, the price for the to connection is multiplied with $\beta = 0.7$, making it in total cheaper than the to connection for the same number of channels. We will see in Section that this cost function will lead to some unexpected results.

Cost per channel	Distance in km <45	<90	⩾90
uv to vv	3.30	4.95	6.60
vv to hv	3.50	5.25	7.00

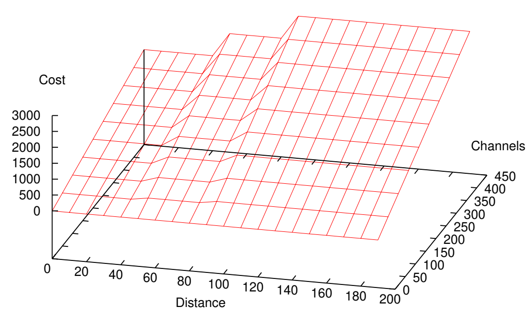


Figure . : Cost function depending on distance and channels

Given the cost function, the question arises which distances to use. In the former projects we always used bee-line distances, since the transport network was not owned by the network operator and not much was known about it. In this project we had the possibility to compute distances in the transport network. Regarding the rationale for the cost function which involved repeaters in the fiber network, this seemed to allow a much better estimate of the involved costs. Further on we will therefore indicate when necessary whether bee-line or transport net distances are used.

16 This is admittedly a strange argument, since usually prices per unit go down with higher capacities. But keep in mind that in case the already installed capacity is not su cient, the placement of new fiber cables is extremely expensive.

4.5.3 Model

Again the model can be derived from the one described in Section . . The nodes in the model are the switching centers. We call the set of all s U, the set of all potential s V and the set of all potential s H. We denote the set of all switching centers by $W = U \cup V \cup H$. Regarding the notation in Section . , we set $V_1 = U$, $V_2 = V$, and $V_3 = H$. While the sets U, V, and H are pairwise disjunctive, the locations associated with the members of the sets may be the same. We introduce a function $\sigma(w), w \in W$ that maps a switching center to its location. If $\sigma(u) = \sigma(v)$ for $v, w \in W$ we call u and v *co-located.* $A_{UV} \subseteq U \times V$ denotes the set of all possible links between s and s, $A_{VH} \subseteq V \times H$ the set of all possible links between s and s. The set of all possible links is denoted by $A = A_{UV} \cup A_{VH}$.

Two types of commodities are used, i. e., $D := \{\text{users}, \text{channels}\}$. Demands $\delta_u^d, u \in U, d \in D$ are only given for s. For each with demands, a co-located with a zero-cost link to the is generated.

We introduce binary variables x_{ij} for each $(i, j) \in A$ indicating active links. Further we have binary variables $y_w, w \in V \cup H$, indicating active s in case $w \in V$, and active s in case $w \in H$. Finally, continuous variables $f_{vh}^d \leqslant \rho_v^d, d \in D, (v, h) \in A_{VH}$ are used to represent the amount of commodity d requested by v from h. The parameter ρ_v^d denotes the maximum capacity of commodity d that can be handled by v. Similarly parameter $\rho_h^d, h \in H$ represents the maximum capacity of commodity d that can be handled by h. This leads to the following model:

$$\sum_{(u,v)\in A_{UV}} x_{uv} = 1 \qquad \text{for all } u \in U \qquad (.)$$

$$\sum_{(v,h)\in A_{VH}} x_{vh} = y_v \qquad \text{for all } v \in V \qquad (.)$$

$$x_{uv} \leqslant y_v \qquad \text{for all } (u,v) \in A_{UV} \qquad (.)$$

$$x_{vh} \leqslant y_h \qquad \text{for all } (v,h) \in A_{VH} \qquad (.)$$

$$\sum_{(u,v)\in A_{UV}} \delta_u^d x_{uv} = \sum_{(v,h)\in A_{VH}} f_{vh}^d \qquad \text{for all } v \in V, d \in D \qquad (.)$$

$$\rho_h^d x_{vh} \geqslant f_{vh}^d \qquad \text{for all } (v,h) \in A_{VH}, d \in D \qquad (.)$$

$$\sum_{vh\in A_{VH}} \beta_v f_{vh}^d \leqslant \rho_h^d \qquad \text{for all } h \in H, d \in D \qquad (.)$$

Constraints (.) to (.) are equivalent to (.) to (.). Equation (.) is a simplification of (.) and (.) is similar to (.). Inequality (.) limits the capacity of the s. Since the utilization of a is dependent on the incoming demands, we have not applied β_v to (.), but to inequality (.) as it reduces the outgoing demands. It is not necessary to explicitly limit the capacity of a , since

$$\sum_{(u,v)\in A_{UV}} \delta_u^d x_{uv} \leqslant \rho_v^d \quad \text{for all } v \in V$$

is implied by (.), (.), (.) and $f_{vh}^d \leqslant \rho_v^d$.

Regarding co-located switching centers two special requirements have to be ensured: If a is active, any co-located has to be connected to it:

$$y_v = x_{uv} \quad \text{for all } (u, v) \in A_{UV} \text{ with } \sigma(u) = \sigma(v)$$

Co-locating a and an is not allowed:

$$y_v + y_h \leqslant 1 \quad \text{for all } v \in V, h \in H \text{ with } \sigma(v) = \sigma(h)$$

It should be noted that in the investigated scenarios all $y_h, h \in H$ were fixed to one, since a reduction of the number of was not considered.

4.5.4 Results

Austria has nine federal states: Burgenland, Carinthia, Lower Austria, Upper Austria, Salzburg, Styria, Tyrol, Vorarlberg, and Vienna. This is reflected in the telecommunication network, since all equipment within each state is from the same manufacturer. An exception is Vienna which has two main switching centers, one from each manufacturer.

The problem can be "naturally" decomposed into four regions which consist of Salzburg and Upper Austria, Tyrol and Vorarlberg, Carinthia and Styria, and as the biggest one Vienna, Burgenland, and Lower Austria. Table . shows the number of switching centers for each region.

Region	HVs	VVs	UVs
Salzburg / Upper Austria	1	12	358
Tyrol / Vorarlberg	1	7	181
Carinthia / Styria	1	9	362
Vienna / Burgenland / Lower Austria	2	15	522

Table . : Size of computational regions

The implementation consists of a set of cooperating programs together with a Web-interface (Figure .). After collecting the user input, the web-interface triggers the other programs and displays the results. First `tapre` a pre-/post-processor for the various input files is started. It generates the input data for Z which in turn generates the mixed integer program which is solved by . After extracting the solution `tapre` is run again to mix the solution with the input and generate result tables and input data for (Wessel and Smith,) which renders maps with the results. Figure . charts the flow of the data. Figures . , . , . , and . show a graphical representation of the results for the respective regions.

is usually[17] able to solve all scenarios to optimality in reasonable time. The only exception was the Vienna, Burgenland and Lower Austria scenario which had an optimality gap of . %. This is far below the accuracy of the data.

17 The solving time for facility location problems depends very much on the cost ratio between connections and facilities. If the ratio is balanced, the problem can get very hard to solve computationally.

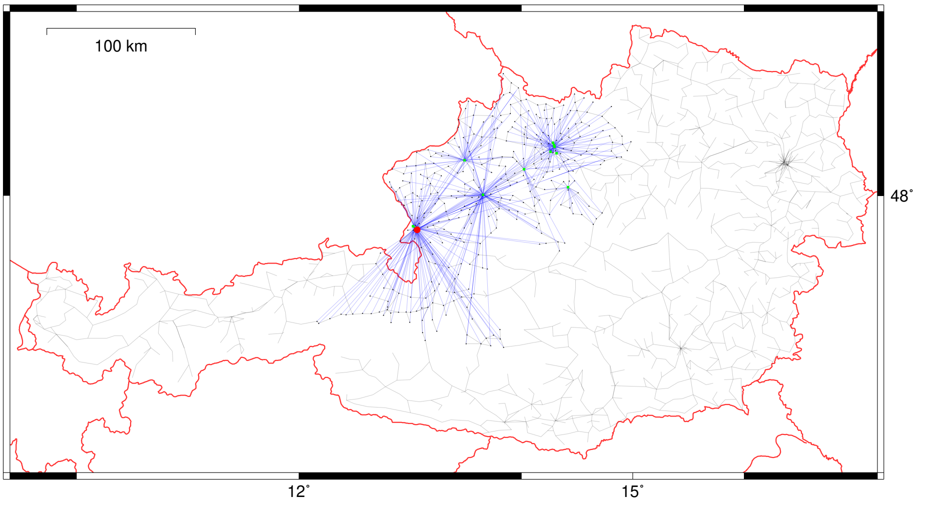


Figure . : Solution for regions Salzburg and Upper Austria

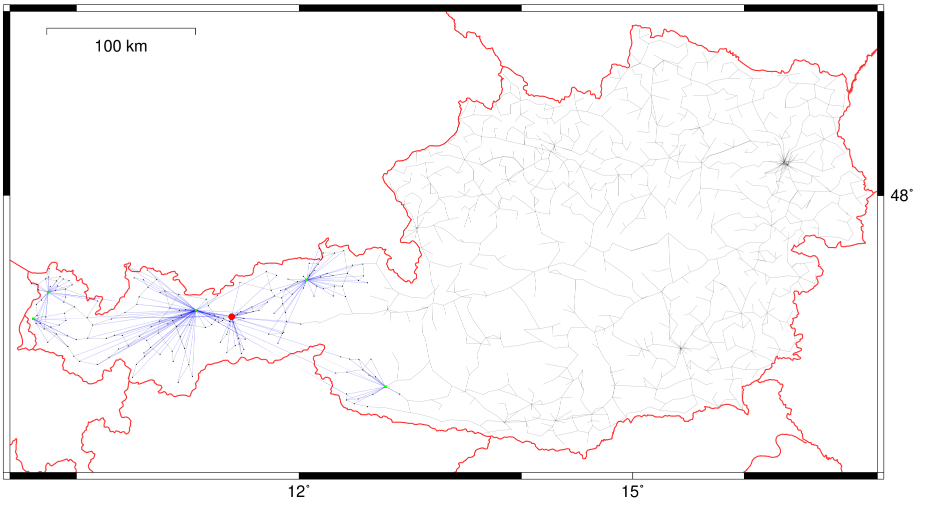


Figure . : Solution for regions Tyrol and Vorarlberg

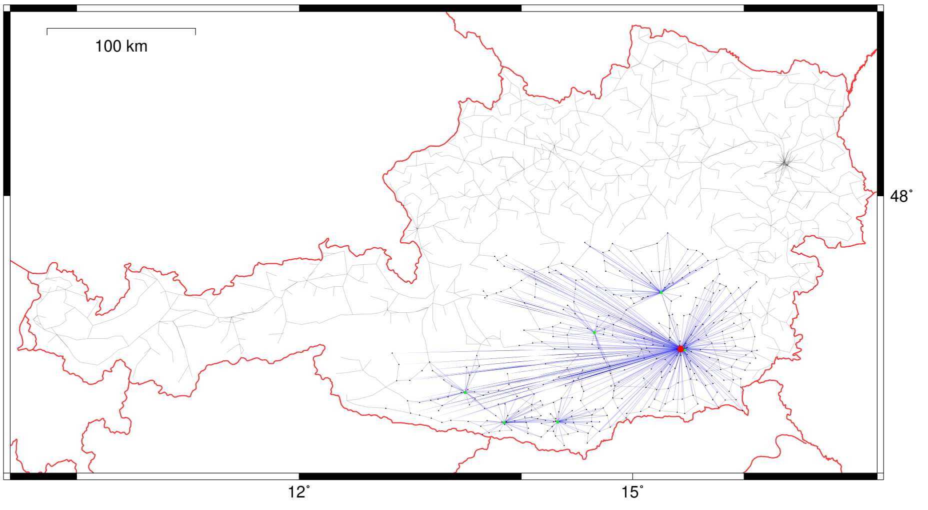


Figure . : Solution for regions Carinthia and Styria

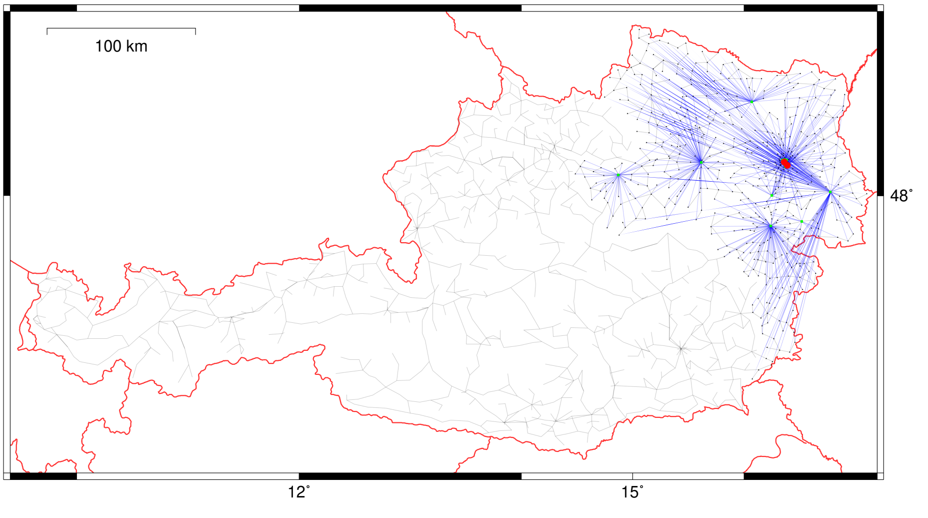


Figure . : Solution for regions Vienna, Burgenland, and Lower Austria

Web-interface
controls

tapre → Z → → awk → tapre →

displays
, ,

Figure . : Program structure

Unexpected results

After solving the scenarios we found that some solutions did not look as expected. We will now present and investigate some cases of apparently strange results. What are possible reasons for unexpected results?

- Anomalies in the data (e. g. without demands)
- Di erences between model and reality
- Reaching of capacity or cost thresholds
- The result is just di erent than expected

Example 1 Looking at Figure . a the following questions can be asked:

- Why is no connected to B?

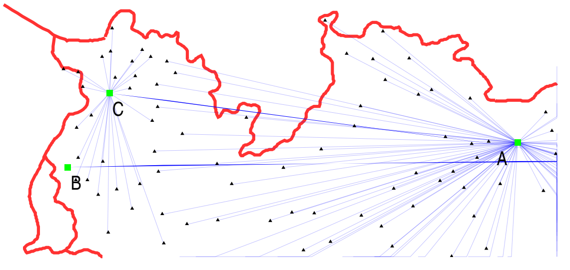
  Since all s in question are less than km away from B and C, the connection costs are equal. Since C has enough capacity all the just happen to be connected to it.

- Why then are B and C both active?

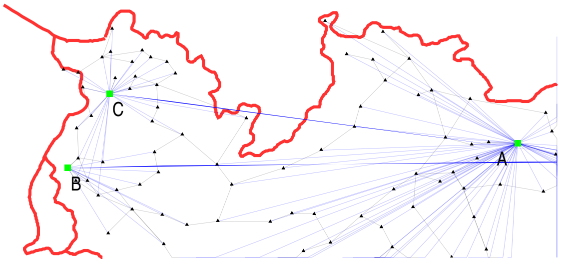
  Because the input data requires B and C to be active.

- Why are some s in the vicinity of C connected to A?

  Because connecting them to C would increase the total length of the link from the to the . to connections are only a little cheaper than to links. So the cost for first connecting to the more remote C does not pay o . As can be seen in Figure . b this changes if instead of bee-line distances transport network distances are used.



(a) Bee-line distances



(b) Transport network distances

Figure . : Results with bee-line vs. transport network distances

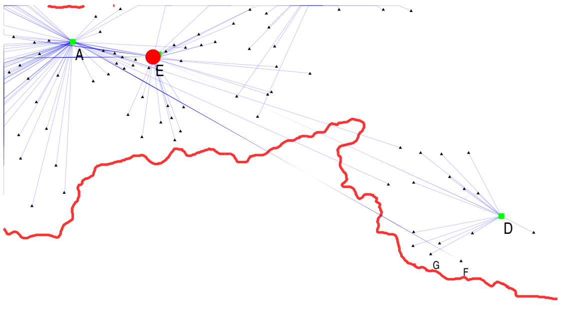


Figure . : Unexpected result with transport network (not shown) distances

Example 2 In this case we use transport network distances. If we look at Figure . (note that E is the green marked "behind" the red), we could again pose some questions about the shown result:

- ▶ Why is F connected to A instead of D?
- ▶ Would not E be better than A to connect F?
- ▶ Why is G connected to D, if F is not?

F demands channels and has the following possibilities to connect to (the costs are taken from Figure . on page):

```
To   Cost         Total
 D   30 * 4.95 = 148.5
 A   30 * 6.60 = 198.0
 E   30 * 6.60 = 198.0
```

Connecting F to D would save . . But the connection cost from A to the would increase (note that due to internal traffic, only channels have to be connected to the) as follows:

```
From  HV-Distance  Cost                  Total
 A       29 km     3.5 * 21 =             73.5
 D      291 km     7.0 * 21 = 147
                            -   49.5   =  97.5
```

So it is obviously cheaper to connect F to A. The connection to E has the same price as connecting to A, because it is also less than km away from the , so the result is arbitrary. But why then is the same not true for G? G demands channels and has the following possibilities:

```
To   Cost         Total
D    26 * 3.3 =   85.8
A    26 * 6.6 = 171.6
E    26 * 6.6 = 171.6
```

Connecting G to D instead of A saves . . But again the connection cost from A to the would increase:

```
From  Cost                     Total
 A    3.5 * 18.2 =              63.7
 D    7.0 * 18.2   = 127.40
                   - 85.8    =  41.6
```

As can be seen, it is still cheaper to connect G to D than to A. Again the cost for E would be identical to A.

Assessment

As we have seen, using stepwise increasing cost functions can be a major source of unexpected results. The same happens if thresholds for capacities are used. Another source of problems are visualizations that do not completely represent the data, as shown here with the transport network distances. It defies the intuition of the observer and makes it considerably more di cult to convince people that the results are useful.

What can the solutions be used for?

- To assess the correctness of assumptions
  (how many switching centers are needed?)
- To compare di erent scenarios
  (what is the impact of capacity changes?)
- To make qualitative cost decisions
  (which switching network is likely to be cheaper?)
- To verify the quality of the model
  (are the assumptions and rules sound?)
- To estimate the potential for savings
  (what would a green-field solution cost?)

What should the solutions not be used for?

- Compute quantitative cost results
- Use of the results without further consideration

Epilogue

Because the hardware is already paid for, only operating costs for the installed hardware occurs. We could show as a result of the project that since refitting switching centers involves additional costs, short-term changes do not pay o .

To the best of our knowledge nobody from the people involved at the start of the project is still working for T A . The department itself was reorganized. It is not known whether the software was ever used again after the installation at T A in .

4.6 Conclusion

We have shown in this chapter how to uniformly handle seemingly di erent problems. Table . gives a summary of the diverse objects that were cast into the same model. As can be seen, none of the projects stretched the abilities of the model to the limit. In fact, most of the time in the projects was spent on assembling, checking, and correcting data, to compile coherent data-sets that fit into the model.

	DFN	e·plus	T	A
V_1 nodes	Client	BSC		UV
V_2 nodes	Backbone	MSC		VV
V_3 nodes	Core			HV
Commodities		subscribers		channels
				users
Configurations		10 per MSC		
Link capacities	discrete			

Table . : Di erent names, same mathematics

We mentioned in the beginning "changing our attitude". It took some time to understand that our foremost task is not to design networks, but to give decision support. In all projects, the networks in question were virtual networks in an existing (mature) infrastructure. It became more and more clear that the precise cost of changes cannot be determined in general and for all possible combinations of changes, if changes on the infrastructure are possible at all. So what we can do is to

- check whether a particular change looks promising,
- find areas which can probably be improved,
- insure decisions, i. e., provide lower bounds, and
- evaluate alternatives.

In our experience regarding facility location problems, real-world data and requirements produce rather restricted problem instances, in the sense that the results are often predictable and that it is hard to find any realistic feasible solution that is much worse than the optimum.

While this sounds as if our work was unnecessary, the opposite is the case. Precisely the fact that the solutions are so inertial shows their usefulness. Given that the data we based our computations on are often only predictions or forecasts and the cost functions are only virtual approximations, highly fluxionary solutions indicate that any decisions based on the results are questionable, because they depend largely on the assumptions made.

The other experience we gained from these projects was that the ability to quickly adapt the model is far more important than to solve the resulting instances to optimality. This insight triggered the use of modeling languages and the development of Z .

Chapter 5

MOMENTUM

A programmer is a person who passes as an exacting expert on the basis of being able to turn out, after innumerable punching, an infinite series of incomprehensive answers calculated with micrometric precisions from vague assumptions based on debatable figures taken from inconclusive documents and carried out on instruments of problematical accuracy by persons of dubious reliability and questionable mentality for the avowed purpose of annoying and confounding a hopelessly defenseless department that was unfortunate enough to ask for the information in the first place.
— Grid news magazine

This chapter introduces the planning problem as it was faced by the work package-4 team in the M[1] *project. I wish to thank all members of the project for the good time and especially the WP-4 team for the collaboration. The results presented in this chapter are joint work with my colleagues Andreas Eisenblätter and Hans-Florian Geerdes.*

5.1 UMTS radio interface planning

The *Universal Mobile Telecommunication System* ( ) is a very complex one. We will describe it in enough detail to explain our endeavor, but will refrain from giving a comprehensive and detailed[2] outline, which can be found in Holma and Toskala ( ).

> *In general terms our task is to put up antennas in such a way that users get service in most places, most of the time.*

Antennas are put up at so-called *sites*, which usually host three or fewer antennas. The antennas used are mostly sectorized, i. e., they have a main direction of radiation and a

1 Project IST- - was funded partly by the European Commission. For more information visit the project's web site at http://momentum.zib.de

2 has numerous possible, but seldom used, configurations and exceptions, which we will only sometimes hint at in footnotes.

beam width. The area in which an antenna is likely to serve mobiles is called its *cell*.

For a user to have service at a given point two things are important: The point needs to be in the coverage area of at least one cell and the cell has to have enough capacity left to match the demands.

This means, the network designer has to answer the following questions: Which sites should be chosen from a limited set of possible candidates? How many antennas should be installed at a specific site[3]? Which types of antennas should be used, and how should the antennas be mounted, i. e., the height, azimuth, and tilt are to be decided, as indicated in Figure . .

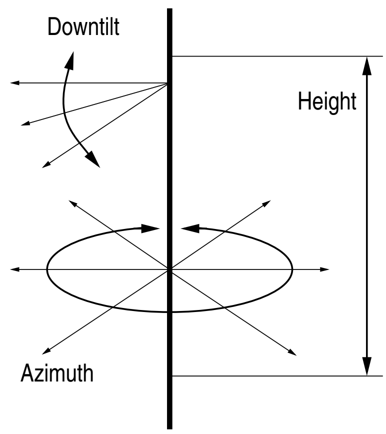


Figure . : Antenna installation

Some of these questions are easy to answer: The usual number of antennas at a site is three[4]. Di erent antenna types are possible, but for maintenance and supplier reasons, the choice is usually restricted. The mounting height is often fixed due to co-location with existing antennas. There are two flavors of tilt, mechanical and electrical. Electrical tilting is preferred, but the amount is restricted depending on the antenna type. Mechanical tilting is apart from constructive obstacles always possible and can be combined with the electrical tilt. This leaves three choices to make:

- Site selection
- Setting the azimuth
- Choosing electrical and mechanical tilt

There are further choices of course, like carriers, primary scrambling codes, processing units, radio-resource-management parameters, etc.[5] But as far as we are concerned, the choices above are those that have the most severe implications for the cost and performance of the network.

These choices are similar to those for systems like . While the answers we give in Section . hold for most radio based telecommunication systems in principle, we will show beginning with Section . what makes planning di erent.

3 In some cases, sites have not only one, but several possible antenna locations, for example, the four corners of a flat roof.

4 All antennas at a site can share the rest of the infrastructure, making it cheaper to have multiple antennas. Omni-directional antennas are seldom used for . Two antennas would have an unfavorable transmission pattern. More than three antennas per site are possible but uncommon.

5 For an explanation of carriers, codes, and radio-resource-management see Holma and Toskala ( ).

In this chapter many e ects on the strength of signals are measured in *Decibel* or dB, which is a dimensionless logarithmic unit often used in telecommunication engineering. A short introduction can be found in Appendix A. on page .

5.2 Coverage or how to predict pathloss

In every antenna emits a constant power *pilot* signal. This is used by the mobiles to measure which antenna is received best. We say a point in the planning area has (pilot) coverage, if and only if the received strength of the pilot signal from at least one antenna is over some threshold. This threshold depends on many factors. For example, to ensure penetration inside buildings an additional loss of about dB has to be assumed on top of the outdoor requirements. The area in which the pilot signal of a specific antenna is received best is called its *best server area*.

In order to compute the signal strength at some point, we have to predict the weakening of the signal. If both sender and receiver are in an open area and have no obstacles between them, this is comparatively easy. Unfortunately, these conditions are seldom met in street canyons and predicting the so-called *pathloss* is an art in itself. The most simple methods to predict pathloss are rules of thumb that include empirical factors according to the target area, like the - model (see, e. g. Kürner,):

$$
\begin{aligned}
L(d, h_e, h_m, f) &= 46.3 + 33.9 \log f - 13.82 \log h_e - a(h_m, f) \\
&\quad +(44.9 - 6.55 \log h_e) \log d + C_m \\
\text{with } a(h_m, f) &= (1.1 \log f - 0.7) h_m - (1.56 \log f - 0.8)
\end{aligned}
$$

where d is the distance between base station and mobile, h_e is the *effective height*[6] of the base station, h_m is the height of the mobile (all heights and distances measured in meters), and f is the frequency of the signal in megahertz. C_m is a correction factor depending on the density of the buildings.

At the other end, prediction models utilizing building data and sophisticated ray tracing methods are used to even incorporate the e ects of refraction on buildings and the like. As a result the quality of the prediction varies widely. Have a look at Figure . (darker areas imply less pathloss[7]). For further information on how to compute pathloss predictions see, e. g., Geng and Wiesbeck ( ), Saunders ( ), Bertoni ( ).

In the pathloss predictions shown, the e ects of a specific antenna are already incorporated. To do this, knowledge of the radiation properties of the antenna is needed. Figure . shows the attenuation of the signal in relation to the horizontal and vertical angle of radiation for a Kathrein[8] antenna.[9]

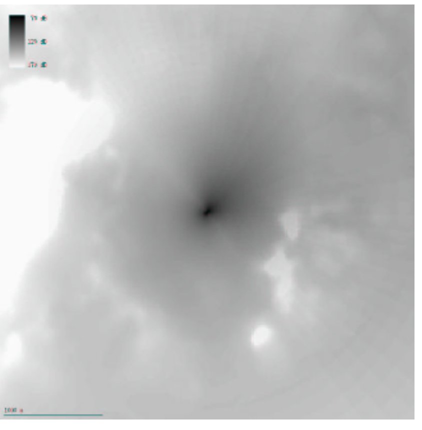
6 There are several ways to compute the e ective height. The simplest one is the so-called *spot-height* which is defined as the di erence between the height of the antenna and the ground height of the mobile.

7 Only prediction . d allows to distinguish between indoor and outdoor areas. For the other predictions it is assumed that the mobile is located outdoors. This is the reason why the prediction in Figure . b looks darker. It has less pathloss on average than . d.

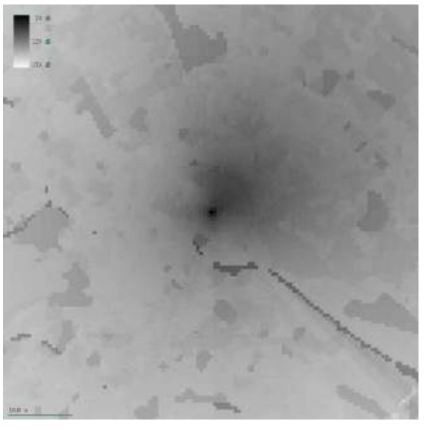
8 http://www.kathrein.de

9 For the interpretation of the diagrams keep in mind that the diagrams are inverted. Antenna patterns

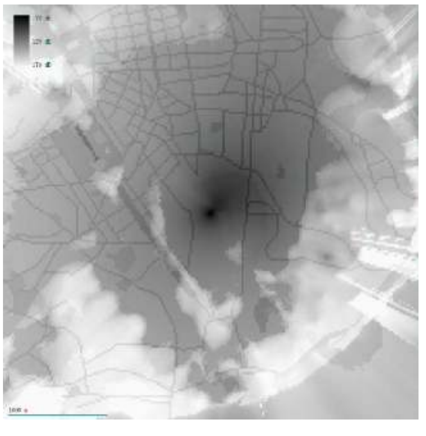
In all predictions shown on this page the same antenna with ° electrical tilt, ° mechanical tilt, and ° azimuth is used.



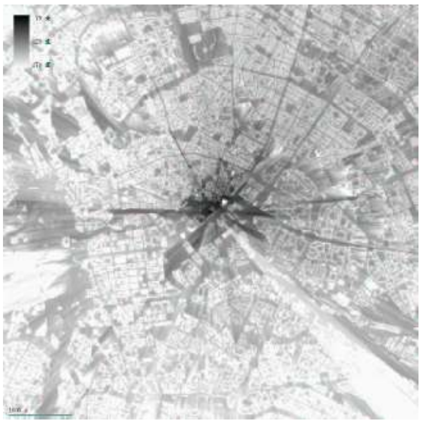
(a) Lisbon, spot-height - predictor with m resolution height data



(b) Berlin, M predictor with m resolution height and clutter data

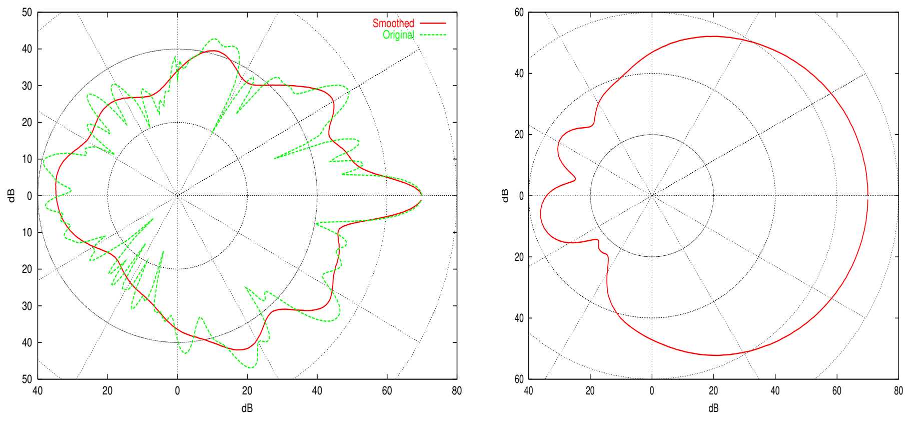


(c) Lisbon, M predictor with m resolution height, clutter and merged vectorized street data



(d) Berlin, e·plus predictor with m resolution height, clutter, vectorized street and building data

Figure . : Pathloss predictions



(a) vertical (b) horizontal

Figure . : Antenna diagrams

Several methods for interpolating the combined vertical and horizontal loss from the vertical and horizontal antenna diagrams are known, see, e. g., Balanis ( ), Gil et al. ( ). A specific problem is the high volatility that can be seen in the vertical diagram (green drawing in Figure . a). Due to reflection, refraction, and spreading it seems highly unlikely that this amount of volatility can be observed in practice. Unfortunately, we were not able to confirm this assumption with measurements. Since using an unsmoothed vertical diagram leads to some strange e ects in the optimization procedures due to (probably artificial) details in the pathloss predictions, we decided to use smoothed diagrams (red drawing in Figure . a).

To obtain the total end-to-end attenuation, we have to subtract some losses due to the wiring and equipment. There is also a so-called *bodyloss* depending on the user, i. e., whether the mobile is attached to the ear or connected to a hands-free speaking system in a car.

We have now covered most of the static aspects that determine the attenuation of the signal. But there are also dynamic factors, like slow and fast fading, moving obstacles, the season[10] and several more. Since we have a static view of the problem and have to make plans regardless of the season, we incorporate these e ects by using additional o sets to change the attenuation to either average or worst case levels.

describe the attenuation of the signal. Straight drawing would lead to the counterintuitive situation that the strongest signal is sent in that direction that is nearest to the origin, i. e., the center of the diagram. So what is drawn is dB minus attenuation level at the specific angle. This leads to the expected interpretation that the radiation is strongest where the points are most distant from the center.

10 Because the trees have di erent influence with and without foliage.

5.2.1 How much freedom do the choices give us?

The question how accurate our pathloss predictions are is quite important, since most planning approaches are based on it. A related question is how much influence our planning choices have on the pathloss. Since we are not able to answer these questions conclusively, we will shed some light on the fact that without calibration from measurements in the target area, the results from pathloss predictions should be taken with a (big) pinch of salt.

Assuming that we have a fixed ° angle between the sectors, the maximum difference in pathloss from changing the direction is about dB (see Figure .). In the M Berlin public scenario (Rakoczi et al., , Geerdes et al.,) the average distance for each site to its nearest neighbor is m. Figure . shows the distribution. Figure . shows the di erence in pathloss resulting from an unsmoothed vertical antenna diagram for the beginning and end of a m pixel in case of an antenna height of m and ° electrical tilt. As can be seen, for a pixel m to m away from the antenna, the variation in pathloss alone from the changing angle to the antenna is about dB.

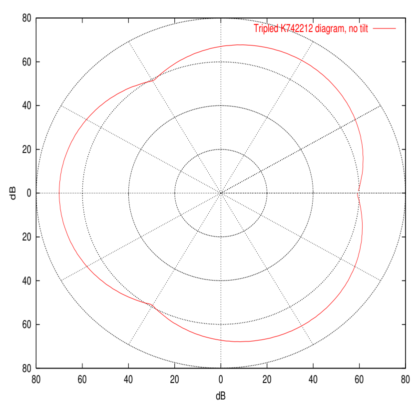


Figure . : Tripled horizontal diagram

Figure . displays the minimum and maximum pathloss we can obtain by changing the azimuth and electrical tilt. We assumed a ° angle between the three sectors and a antenna at m height. The antenna allows between zero and eight degrees of electrical tilt. The maximal pathloss di erence is about dB at m, which means that changing the tilt gives us another dB in addition to the azimuth variations. Also drawn is the isotropic - prediction, i. e., without taking the antenna into account. Figures . and . visualize the change in pathloss due to changes in electrical tilt and height, respectively.

We will come back to this topic in Section . . , once we have introduced the most important concepts in .

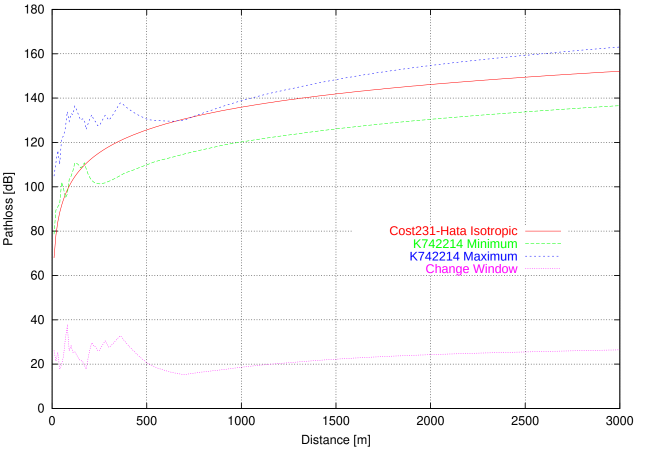


Figure . :
Pathloss change depending on distance (min/max)

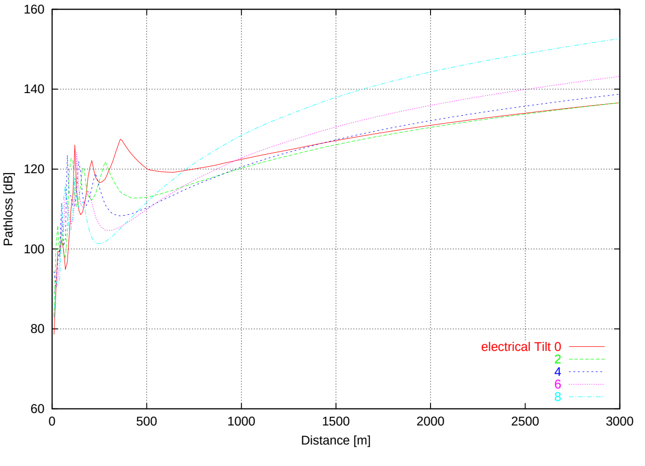


Figure . :
Pathloss change depending on tilt

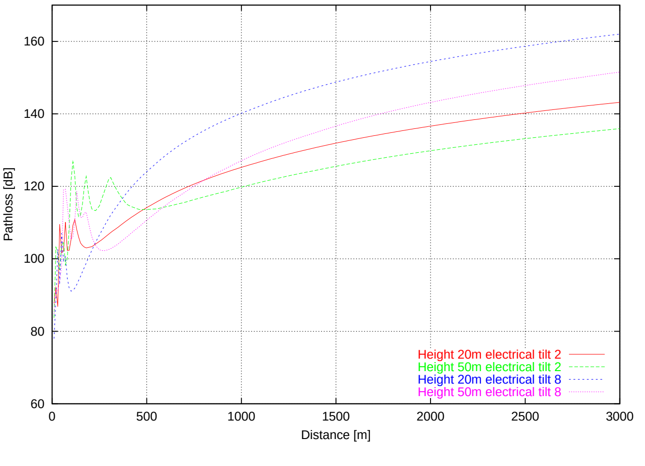


Figure . :
Pathloss change depending on height and tilt

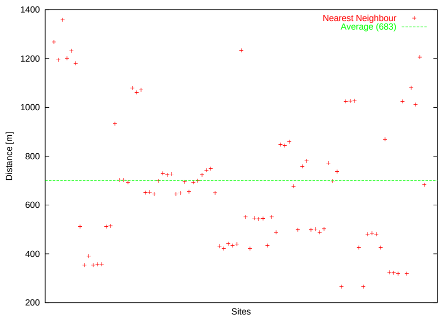


Figure . : Site distances in Berlin

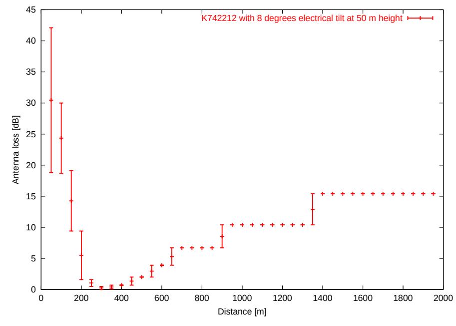


Figure . : Discretization jitter

5.3 Capacity or how to cope with interference

Up to this point we have not talked about how works. is a so-called *Wideband Code Division Multiple Access* ( ) system. Wideband means instead of having each transmitter using its own small frequency band, all transmitters use the same broad band (carrier). There is no partitioning of the available frequency band into several channels. All transmitters send on the same frequency band at the same time. To make this work, a signal that usually could be send on a kilohertz wide band is spread to, say, five megahertz. This is done in a particular way[11] that allows the receiver to separate this spreaded signal out of all the others provided that one condition is satisfied:

> *The ratio of the strength of the signal to separate against all the other signals that interfere must exceed a specific threshold.*

This means we have to know three things:

- *How strong is the received signal?*
- *How strong is the total interference?*
- *Which signal-to-interference ratio is needed to receive data at a certain rate?*

A few things to note:

- The needed ratio depends on the data rate of the transmission, because the higher the data rate is, the more bandwidth is needed, and the less we can spread the signal making it more di cult to separate.
- If we send a strong signal, somebody else receives strong interference.
- While the power of the pilot signal is constant, the powers from the mobile to the base-station and vice-versa are adjusted times per second. This is called *Fast Power Control* and is done to maintain a power level just above the required threshold, while compensating for changes in the received signal strength due to all kind of e ects like movement, change in interference, etc.

11 For all the details see Holma and Toskala ( )

We have to distinguish two situations: Transmissions from the mobile to the base-station, the so-called *uplink*, and transmissions from the base-station to the mobile, or *downlink*. Since is mostly used in *Frequency Division Duplex* ( ) mode, uplink and downlink have separate frequency bands, i. e., they do not interfere with each other. We assume that whenever a mobile is connected, it sends to and receives from a single specific antenna. In reality, the situation is more complicated, since a mobile can be linked to two or more antennas, a state called *soft-handover*[12].

5.3.1 The CIR inequality

We call a fixed setting consisting of antenna type, height, azimuth, electrical, and mechanical tilt at a specific site an *installation*. The set of all considered (possible) installations is denoted $\mathcal{I}$. Members of this set are denoted by i.

Unlike pilot coverage, interference cannot be viewed independently of the tra c. Assuming some given tra c distribution, we have tra c realizations in form of a set $\mathcal{D}$ of *snapshots*. We denote members of $\mathcal{D}$ by d. Each snapshot consists of a set of mobiles $\mathcal{M}_d$. The union of all mobiles is $\mathcal{M} := \cup_{d \in \mathcal{D}} \mathcal{M}_d$ and its members are denoted by m.

If we look at a specific mobile m, we do not expect it to send all the time. Depending on the service used, the *transmit activity factor* di ers. For example, when doing voice telephony usually a factor of % is assumed, meaning that on average only one of the two people involved speaks at the same time. With other services like or video streaming this factor can be highly asymmetric between uplink and downlink. We denote the transmit activity factor of a mobile by $\alpha_m^{\uparrow}$ for the uplink and $\alpha_m^{\downarrow}$ for the downlink.

The end-to-end attenuation[13] between a mobile and an installation including all o sets as described in the previous section is denoted by $\gamma_{mi}^{\uparrow}$ for the uplink and by $\gamma_{im}^{\downarrow}$ for the downlink. As mentioned before, each mobile must reach a certain signal-to-interference ratio in order to get service. For technical reasons this is called the *Carrier-to-Interference Ratio* ( ) target and denoted by $\mu_m^{\uparrow}$ in uplink and by $\mu_m^{\downarrow}$ in downlink.

The transmission power of a mobile m in uplink is denoted by $p_m^{\uparrow}$. The received signal strength at installation i is then $\gamma_{mi}^{\uparrow} p_m^{\uparrow}$. If we denote the received background noise at installation i by η_i, the complete inequality for the uplink transmission for a mobile $m \in \mathcal{M}_d$ to installation $i \in \mathcal{I}$ reads:

$$\frac{\gamma_{mi}^{\uparrow} p_m^{\uparrow}}{\eta_i + \sum_{\substack{n \neq m \\ n \in \mathcal{M}_d}} \gamma_{ni}^{\uparrow} \alpha_n^{\uparrow} p_n^{\uparrow}} \geqslant \mu_m^{\uparrow} \quad .$$

Writing

$$\bar{p}_i^{\uparrow} := \eta_i + \sum_{m \in \mathcal{M}_d} \gamma_{mi}^{\uparrow} \alpha_m^{\uparrow} p_m^{\uparrow}$$

12 If all involved antennas are at the same base-station this is called *softer-handover*. Combinations of soft-handover and softer-handover can also occur.

13 Pathloss is in principle symmetric. Due to di erent equipment in uplink and downlink, e. g. mast-head amplifiers, the end-to-end attenuation can be asymmetric.

for the *average* total received power at installation i, this simplifies to

$$\frac{\gamma^{\uparrow}_{mi}p^{\uparrow}_{m}}{\bar{p}^{\uparrow}_{i} - \gamma^{\uparrow}_{mi}\alpha^{\uparrow}_{m}p^{\uparrow}_{m}} \geqslant \mu^{\uparrow}_{m} \quad . \qquad (\ . \ )$$

The downlink case is a little more complicated, because we have two additional factors to include. First of all, we have to take into account the pilot and common channels, the transmission power of which we denote by $\hat{p}^{\downarrow}_{i}$ and $\check{p}^{\downarrow}_{i}$. Another feature we have not yet considered is downlink orthogonality. Each antenna selects orthogonal transmission codes[14] for the mobiles it serves, which then in theory do not mutually interfere. However, due to reflections on the way, the signals partly lose this property, and signals from other antennas do not have it at all. So, when summing up the interference, the signals from the same cell are cushioned by an environment dependent *orthogonality factor* $\bar{\omega}_{im} \in [0, 1]$, with $\bar{\omega}_{im} = 0$ meaning perfect orthogonality and $\bar{\omega}_{im} = 1$ meaning no orthogonality. For notational convenience the interference from other transmissions is denoted by

$$\phi(m, i) = \sum_{\substack{n \in \mathcal{M}_d \\ n \neq m}} \Big(\overbrace{\bar{\omega}_{im}\gamma^{\downarrow}_{im}\alpha^{\downarrow}_{n}p^{\downarrow}_{in}}^{\text{from same cell}} + \overbrace{\sum_{\substack{j \in \mathcal{I} \\ j \neq i}} \gamma^{\downarrow}_{jm}\alpha^{\downarrow}_{n}p^{\downarrow}_{jn}}^{\text{from other cells}} \Big) \quad .$$

Let $\hat{\phi}(m, i)$ denote the interference from other pilot signals and $\check{\phi}(m, i)$ denote the interference from the other common channels:

$$\hat{\phi}(m, i) = \sum_{\substack{j \in \mathcal{I} \\ j \neq i}} \gamma^{\downarrow}_{jm}\hat{p}^{\downarrow}_{j} \qquad \check{\phi}(m, i) = \sum_{\substack{j \in \mathcal{I} \\ j \neq i}} \gamma^{\downarrow}_{jm}\check{p}^{\downarrow}_{j}$$

Writing η_m for the noise value at mobile m, the formula for the downlink reads:

$$\frac{\gamma^{\downarrow}_{im}p^{\downarrow}_{im}}{\phi(m, i) + \underbrace{\bar{\omega}_{im}\gamma^{\downarrow}_{im}\hat{p}^{\downarrow}_{i}}_{\text{own pilot signal}} + \hat{\phi}(m, i) + \check{\phi}(m, i) + \eta_m} \geqslant \mu^{\downarrow}_{m}$$

Defining the total *average* output power of installation i as

$$\bar{p}^{\downarrow}_{i} := \sum_{m \in \mathcal{M}_d} \alpha^{\downarrow}_{m}p^{\downarrow}_{im} + \hat{p}^{\downarrow}_{i} + \check{p}^{\downarrow}_{i} \quad ,$$

we obtain the downlink version of the equation for the transmission from installation $i \in \mathcal{I}$ to $m \in \mathcal{M}_d$:

$$\frac{\gamma^{\downarrow}_{im}p^{\downarrow}_{im}}{\bar{\omega}_{im}\gamma^{\downarrow}_{im}\left(\bar{p}^{\downarrow}_{i} - \alpha^{\downarrow}_{m}p^{\downarrow}_{im}\right) + \sum_{j \neq i}\gamma^{\downarrow}_{jm}\bar{p}^{\downarrow}_{j} + \eta_m} \geqslant \mu^{\downarrow}_{m} \qquad (\ . \ )$$

The inequalities (.) and (.) are central to understand . There is a similar constraint for the pilot signal.

14 The number of these codes is limited. A base can run out of codes. In this case codes from a second set (code tree) are used which are not completely orthogonal.

5.3.2 Assumptions and simplifications

Next, we will try to give some insight into the consequences arising from (.) and (.). In order to do so, some assumptions are made that do not hold for a live dynamic , but facilitate a more deterministic examination.

As noted before, the power from and to a mobile is adjusted times a second in . The bias in the control loops is to lower the power as much as possible, while still ensuring service. From now on, we will assume *Perfect Power Control*, i. e., the transmission powers are at the minimum possible level (see for example Bambos et al.,). It is to be expected that a real system will work on average on a higher transmission power level (see Sipilä et al.,).[15]

We expect that with measurements from live networks it will be possible to find suitable o sets to correct any models based on this assumption.

If we assume perfect power control it follows that inequalities (.) and (.) are met with equality, i. e., the inequalities become equations. In reality, this might sometimes be impossible, for example, due to required minimum transmission powers.

We also assume that if a mobile is served, this is done by its *best-server*, which is the antenna whose pilot signal is received with the highest signal strength.[16] In reality, this is only the case with a certain probability.[17]

Finally, we neglect soft-handover. This is a situation where two or more equally strong, or in this case weak, antennas serve a mobile. The Radio Network Controller can choose the best received signal in uplink and the mobile can combine the transmission power of all antennas in downlink. The main benefit is a more seamless handover of a moving mobile from one cell to another. The drawback is higher interference and therefore reduced capacity in the downlink.

5.3.3 Cell load

Assuming equality, we can deduce how much of the capacity or transmission power of an antenna is used by a specific mobile from (.) and (.). In uplink we can rearrange (.) to yield $p^{\uparrow}_{m}$ depending on the total power:

$$p^{\uparrow}_{m} = \frac{1}{\gamma^{\uparrow}_{mi}} \cdot \frac{\mu^{\uparrow}_{m}}{1 + \alpha^{\uparrow}_{m}\mu^{\uparrow}_{m}} \cdot \bar{p}^{\uparrow}_{i}$$

15 Even though there are several engineering provisions to ensure an acceptable behavior of the system, there is, in fact, no proof that the system will find power levels anywhere near the optimum. At least theoretically the system might even start to swing.

16 If all antennas send their pilots with the same power, the best-server is equal to the antenna which has the smallest pathloss to the mobile. If the transmission power of the pilot signal varies between antennas, the situation occurs that the best-server is not the antenna that is received best. This can have problematic repercussions.

17 Due to obstacles, for example, the best-server for a moving mobile might frequently change. To smoothen the dynamics in the systems, hysteresis is used in many cases. The Radio Network Controller might also decide to connect a mobile to a di erent base-station for load balancing.

The received power from mobile m at installation i is on average $\alpha_m^{\uparrow}\gamma_{mi}^{\uparrow}p_m^{\uparrow}$. Writing this as a ratio of the total received power at the installation we get the *uplink user load*:

$$l_m^{\uparrow} := \frac{\alpha_m^{\uparrow}\gamma_{mi}^{\uparrow}p_m^{\uparrow}}{\bar{p}_i^{\uparrow}} = \frac{\alpha_m^{\uparrow}\mu_m^{\uparrow}}{1+\alpha_m^{\uparrow}\mu_m^{\uparrow}} \qquad (.)$$

$l_m^{\uparrow} \in [0,1[$ is the fraction of the total power received at installation i that is caused by mobile m. Note that the uplink user load is completely independent of the attenuation of the signal.

In the downlink case, we basically repeat what has just been done for the uplink. The starting point is the constraint (.), again assuming that the constraint is met with equality for all mobiles, it can be rewritten as:

$$\frac{1+\bar{\omega}_{im}\alpha_m^{\downarrow}\mu_m^{\downarrow}}{\alpha_m^{\downarrow}\mu_m^{\downarrow}}\alpha_m^{\downarrow}p_{im}^{\downarrow} = \bar{\omega}_{im}\bar{p}_i^{\downarrow} + \sum_{j\neq i}\frac{\gamma_{jm}^{\downarrow}}{\gamma_{im}^{\downarrow}}\bar{p}_j^{\downarrow} + \frac{\eta_m}{\gamma_{im}^{\downarrow}}$$

We define the *downlink user load* of serving mobile m as:

$$l_m^{\downarrow} := \frac{\alpha_m^{\downarrow}p_{im}^{\downarrow}}{\bar{\omega}_{im}\bar{p}_i^{\downarrow} + \sum_{j\neq i}\frac{\gamma_{jm}^{\downarrow}}{\gamma_{im}^{\downarrow}}\bar{p}_j^{\downarrow} + \frac{\eta_m}{\gamma_{im}^{\downarrow}}} = \frac{\alpha_m^{\downarrow}\mu_m^{\downarrow}}{1+\bar{\omega}_{im}\alpha_m^{\downarrow}\mu_m^{\downarrow}} \qquad (.)$$

5.3.4 Pathloss revisited

With the knowledge gained, we can again ask about the precision of our pathloss predictions. Figure . shows the comparison between a m resolution - and a m resolution prediction. Two sites from the M Berlin scenario with a distance of m were chosen. All shown predictions are isotropic, i. e., without antenna e ects. Figures . a and . b show predictions computed by e·plus. Figures . d and . e show - predictions of the same sites. Figures . c and . f are best server maps of the area based on the respective predictions. If the pathloss di erence is less than three decibel, then the area is colored white. Note that since we are working in dB, i. e., on a logarithmic scale, di erence means ratio.

	Site 1	Fig.	Site 2	Fig.	Diff.	Fig.
Average pathloss						
5 m 3D prediction (dB)	132.0	5.10a	126.1	5.10b		
50 m COST231-HATA (dB)	132.3	5.10d	129.5	5.10e		
Correlation	0.77		0.81		0.87	
Average difference (dB)	1.49	5.10g	6.48	5.10h	1.83	5.10i
Standard deviation	10.28	5.10j	9.65	5.10k	7.01	5.10l

Table . : Comparison of - and pathloss predictions

Figure . g visualizes the di erence between the two prediction methods for the first site and . h for the second one. Areas in which the high resolution predictor predicted less pathloss than the - predictor are marked red. In the opposite case the area is marked green. Areas where both predictions do not di er by more than three decibel are colored white, indicating concordance between the two predictions. Figures . j and . k show the respective di erence histograms. Note that the 0 dB bar is clipped and corresponds in both cases to %.

Table . shows some statistical data about the comparison. The data indicates that even though - is only a very simple prediction method, at least for the flat urban area of Berlin, it is a suitable method to compute the average pathloss quite accurately. On the other hand, it is clear that the particular pathloss at a specific point can and usually will vary widely around this average.

Figure . i and the accompanying histogram . l show what we are really concerned about: Where would we have di erences in the ratio between the pathloss from the two sites if we use a - prediction instead of a high resolution prediction? White areas indicate that the ratios between the two predictors di er by less than three decibel. Note that the pathloss di erence between the two antennas was clipped at dB.[18] Red indicates areas where the - predictor sees the ratio between the sites more in favor of the second site than the predictor. Green areas indicate the opposite. Note that the 0 dB bar in Figure . l is clipped and corresponds to %.

As can be seen in Table . , statistically the di erence between the predictions and - is in fact smaller for the pathloss ratio than for the pathloss values itself.[19]

5.3.5 Assessment

The sophisticated high resolution predictions showed a standard deviation of about dB in comparison to pathloss measurements performed by e·plus. Since we have no access to the measurement data, we cannot assess the quality of the - predictions against the real world. But with the m - predictions essentially averaging the m predictions with a standard deviation of about dB we can expect them to have only a slightly higher deviation against the measurements for m resolution.

If we recall Figures . to . we can see that within a radius of about m around an antenna the pathloss is highly volatile. This might not be too problematic, because as Figure . shows, the absolute value will be on average high enough to ensure coverage and, as Figure . l indicates, the area is likely to be dominated by the local antenna. On the other hand, this is probably all we can say with some certainty about this area. It should be noted that - was not meant to be used for distances below km. If we are at least that far away, we can expect[20] to get a fair idea of the average pathloss

18 A factor of dB means that one of the signals will be attenuated times more than the other. If the di erence is this big, then it is unlikely to matter in interference calculations. Therefore, in this picture white, meaning "no di erence", is also used if the di erence in pathloss between the two antennas for both prediction methods is bigger than dB, regardless of the actual amount.

19 This result is slightly misleading, as Sipilä et al. ( ) show from simulations that due to e ects resulting from multi-path reception and fast power control the observed interference will be higher, at least for slow moving mobiles.

20 Our conclusions so far are drawn from one arbitrarily chosen example.

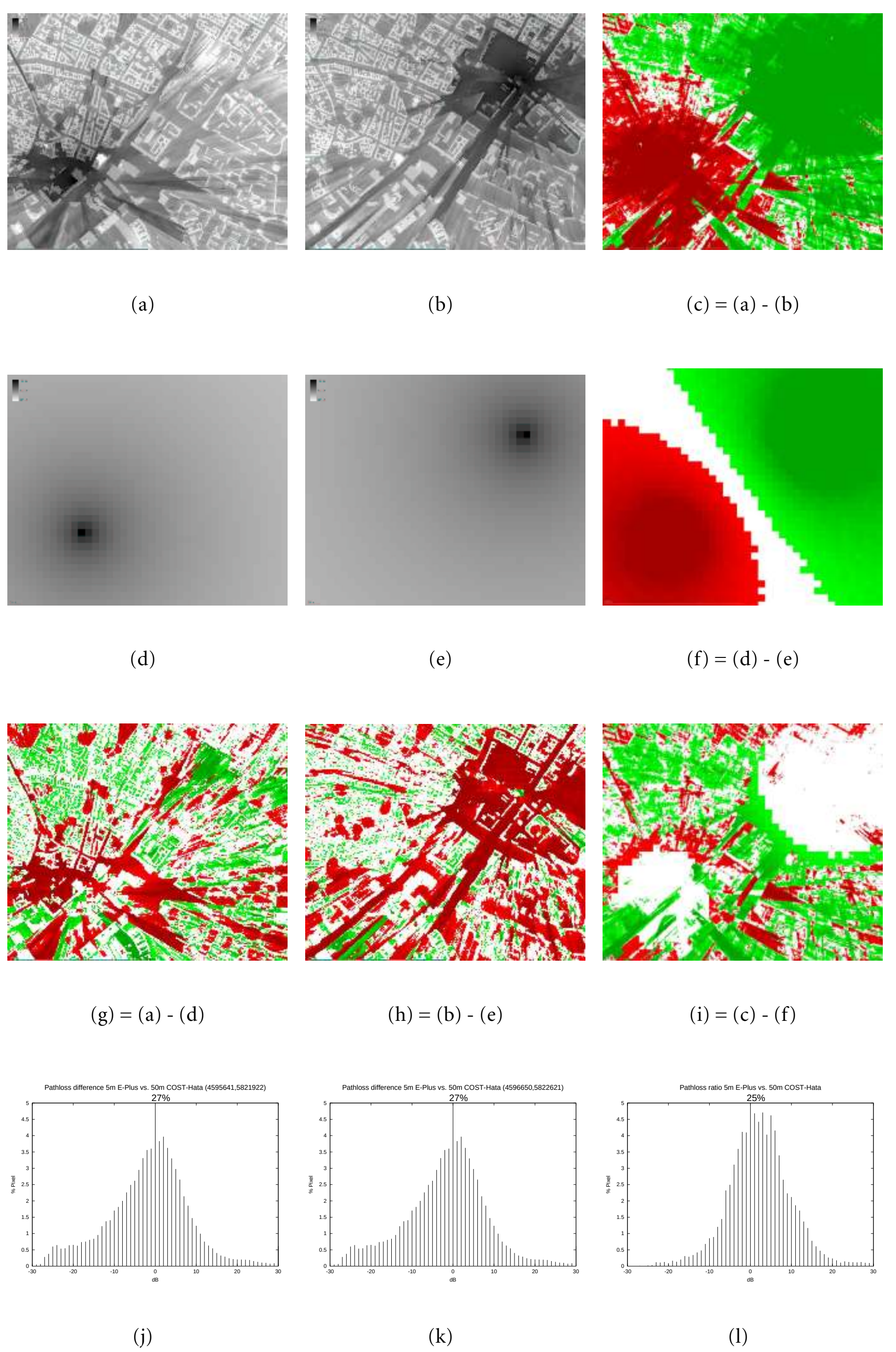


Figure . : Comparison of - and pathloss predictions and especially pathloss ratios between di erent sites from the predictions.

To assess the implications of this insight, we need to have an idea on how the possible scenarios look like. This can be split into two parts. The question is whether we are

- coverage or capacity limited
- uplink or downlink limited

In rural areas with low tra c, coverage will be the limiting factor. In dense urban areas with high tra c, we expect to be capacity limited. All services o ered for so far have either symmetric tra c demands, e. g., voice or video telephony, or if they are asymmetric they have the higher demand in downlink, e. g., video streaming, , E-mail.[21] Hence, we assume that especially in high tra c areas, the downlink will be much more demanding. Since deployment will start in urban areas, we expect the downlink capacity limited scenario to be the most important.

Looking at Figure . we see that in a realistic urban scenario the nearest neighbor of most sites is less than km away. In many cases two sites are less than m apart. The standard deviation of the prediction data is about as high as the maximum change we can achieve by either turning or tilting antennas. Adding the fact that all tra c forecasts especially for the new services are educated guesses at best, we can conclude that qualitative results on average behavior of systems seem possible while any quantitative results based on this kind of data should be very carefully assessed and viewed with suspicion.

5.4 Models

Now that we have a little understanding of how works, we will present three optimization models to address planning choices. We will use models based on set covering and partitioning (see, e. g., Borndörfer,) for the site and azimuth selection.

5.4.1 Site selection

As we stated in the beginning, the first task is to select sites. The planning area is discretized, i. e., divided into square pixels of some resolution, usually between m and m. Each point in a pixel shares all the attributes, like pathloss, with all other points in this pixel.

Given a set of possible sites $\mathcal{S}$ and a set of pixels P, we compute *cover-sets* $S_p \subseteq \mathcal{S}, p \in P$ which contain all sites that can possibly serve pixel p. Since we have not yet decided on azimuth and tilt of the antennas, and because the capacity of a cell depends on its neighbors, it is not possible to know precisely which pixel can and will be served by which site. Therefore, we have to build the cover-sets heuristically, along the following ideas for example:

21 As of June , all providers o ering services are limited to kbit/s in uplink vs. kbit/s in downlink.

- Include only pixels, whose predicted pathloss (either isotropic, minimum possible, complete including antenna) is above a threshold (coverage).
- Use the cell load arguments from Section . . , to sum up the load using the average tra c on the pixel until some threshold is exceeded (capacity).
- Ignore pixels with a distance greater than a certain threshold.
- If the potentially served areas due to the coverage criterion are not coherent, use only those adherent to the site location.

Have a look at Figure . a. For a threshold of dB the green area will have coverage and is connected to the center. Areas marked in red are enclosed by the green area, but will not have su cient coverage. Areas which will su er from heavy interference are marked blue.

A related question is, which pixels to include into the selection at all. It might be a good idea to ignore pixels that have no tra c, that are on the border of the scenario, or even some of those that can only be served by a single site, since this would mean the site is surely selected.

We introduce binary variables $x_s, s \in \mathcal{S}$ and $z_{ij}, i, j \in \mathcal{S}$. $x_s = 1$ if and only if site s is selected and $z_{ij} = 1$ if and only if sites i and j are both selected. The latter gives us some leverage to minimize interference using a heuristical penalty factor $0 \leqslant c_{ij} \leqslant 1$ indicating the amount of interference between sites i and j.

Our objective is to minimize the number of sites and the interference:

$$\min \sum_{s \in \mathcal{S}} x_s + \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{S}} c_{ij} z_{ij} \quad .$$

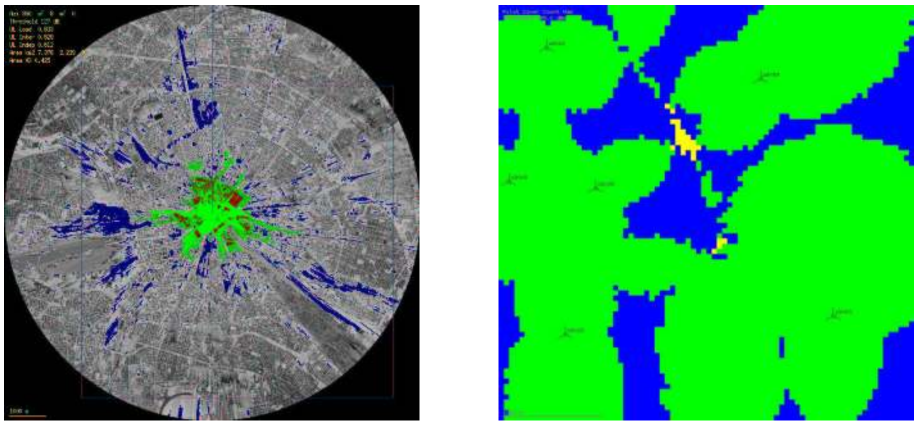
The constraints are:

$$\sum_{s \in S_p} x_s \geqslant 1 \qquad \text{for all } p \in P \text{ with } S_p \neq \emptyset \qquad (.)$$

$$x_i + x_j - z_{ij} \leqslant 1 \qquad \text{for all } i \in \mathcal{S}, j \in \mathcal{S} \qquad (.)$$

(.) ensures that each pixel can be served, and (.) marks sites that mutually interfere. (.) is the typical formulation for the logical $a = b \wedge c$ as shown in Equation . . The first two parts of . can be omitted, because we are minimizing and $c_{ij} \geqslant 0$ in all cases.

The solvability of the resulting is empirically good. We have successfully used it for commercial scenarios with more than sites. Figure . b shows a solution of this model for the M The Hague public scenario (Rakoczi et al., , Geerdes et al.,). Three antennas with ° angles were installed at each selected site. Green and blue areas have su cient pilot coverage. Blue areas are covered by a single installation. An example of the model formulated in Z can be found in Appendix C. on page .



(a) Berlin selection criteria

(b) Pilot count

Figure . : Site selection

5.4.2 Azimuth (and a little tilting)

Jakl et al. ( ) gave some ideas how to analytically[22] derive settings for the azimuth and tilt. We will show how to quickly transform some of these ideas into s.

According to Nawrocki and Wieckowski ( ), the optimal azimuth setting for antennas in a hexagonal grid looks like Figure . . The idea is that each antenna should point directly to another site and that the angles between the beam directions of the antennas should be maximized. Additionally, the number of beams crossing each other should be minimized.

We start by building a directed simple graph $G = (S, A)$ with the site set S being the set of nodes, and A being the arcs. We include only arcs between sites that are no more than d meters apart. We choose a set packing approach and introduce binary variables $x_{ij} = 1$, if and only if arc $(i, j) \in A$ is selected, i. e., if an antenna is pointing from site i to site j.

Additionally, we set binary variables $y_{ij}^{mn} = 1$, $(i, j) \in A$, $(m, n) \in A$, if and only if antennas pointing from site i to j and from m to n are both active. We only use combinations of i, j, m, n, where at least two of them are equal, i. e., mostly three sites are involved, and the angle between the two arcs is less than °. We denote the set of these arc pairs by $F \subset A \times A$.

Finally, binary variables z_{ij}^{mn}, $(i, j) \in A$, $(m, n) \in A$ are defined. $z_{ij}^{mn} = 1$ if and only if arcs (i, j) and (m, n) are both selected and cross each other. The set of crossing arcs is called $C \subset A \times A$.

22 i. e., without doing explicit or implicit simulations, meaning without looking specifically at the in-equalities.

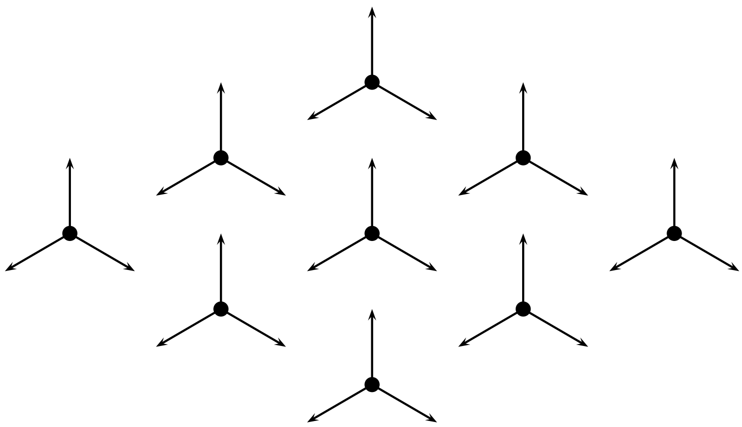


Figure . : Optimal azimuth in a regular hexagonal grid

The objective is to minimize

$$\sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{(i,j,m,n)\in F} v_{ij}^{mn}y_{ij}^{mn} + \sum_{(i,j,m,n)\in C} w_{ij}^{mn}z_{ij}^{mn}$$

where c_{ij} is the cost of arc (i,j), which depends on the pathloss between sites i and j. The heuristic idea is that a high pathloss between the sites is beneficial, since there will be less interference. v_{ij}^{mn} is the penalty for arcs with an angle of less than °. The penalty should increase nonlinearly towards smaller angles. Furthermore, we have w_{ij}^{mn}, the penalty for crossing arcs.

The number of antennas at each site s is fixed in advance and denoted by ζ_s. This gives us a cardinality constraint:

$$\sum_{(s,r)\in A} x_{sr} = \zeta_s \quad \text{for all } s \in S$$

ζ_s is usually set to three. If the angle between two adjacent arcs emitting from a site is greater than ° we mark the site as belonging to the border of the scenario. For these sites we set $\zeta_s = 0$. Using a reduced value like one or two is also possible. Since anti-parallel arcs are interference-wise the worst case, they are not allowed:

$$x_{ij} + x_{ji} \leqslant 1 \quad \text{for all } i,j \in A, i \neq j$$

In order to trigger the setting of the y and z variables we have again reduced logical '$\wedge$' constraints of type (.):

$$\begin{aligned} x_{ij} + x_{mn} - y_{ij}^{mn} &\leqslant 1 \qquad \text{for all } i,j,m,n \in F \\ x_{ij} + x_{mn} - z_{ij}^{mn} &\leqslant 1 \qquad \text{for all } i,j,m,n \in C \quad . \end{aligned}$$

Table . lists some data about the models. Berlin is the original network given with the M Berlin public scenario. Berlin* is a network computed with the site selection model from the previous section. Lisbon is the original network from M - Lisbon public scenario (Rakoczi et al., , Geerdes et al.,). Figure . shows the azimuth optimization results for Berlin* and Lisbon. The blue lines indicate possible antenna directions, the red lines are those present in the solution. Also visible is the most eminent problem of this approach: Since we have restricted the possible directions, sometimes there is simply no good choice available. Introducing artificial sites to increase the number of possible locations could be a remedy for this problem. On the other hand, this is just a heuristic in order to get some sensible directions to begin with. In our experience, the results obtained are a good starting point for a local search heuristic that cleans up "local problems" afterwards.

	Berlin	Berlin*	Lisbon
max. distance [m]	2,200	2,200	1,800
Sites	50	44	52
Cells	111	90	102
Arcs	622	356	1,098
Variables	21,872	4,512	89,858
Constraints	64,011	12,690	266,881
Non-zeros	149,794	29,804	623,516
Optimality gap	6%	2%	12%
Time [h]	2	1	14

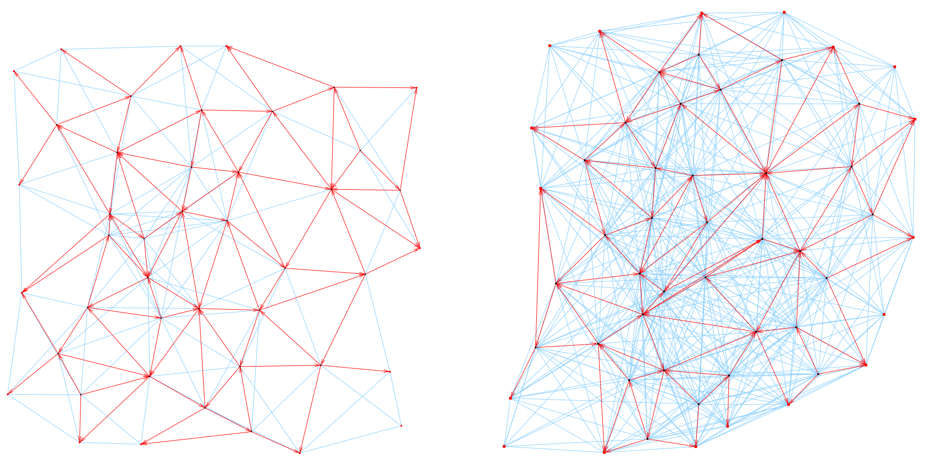
Table . : Azimuth optimization

A Z formulation of the model can be found in Appendix C. on page .

Analytical tilting

Having computed the azimuth, the question of tilting remains. Assuming an antenna installed at site i pointing at site j, we can compute a suitable tilting angle by $\theta = \arctan \frac{h_i}{\lambda d_{ij}}$, with h_i being the height of site i and d_{ij} being the distance between sites i and j. λ is a factor in the range $[0, 1]$ indicating where between i and j the main lope of the antenna should point. This idea can be improved in several ways. One option is to take not only the height of site i, but of the whole area into account.

Having computed an angle θ, we first try to accomplish this by electrical tilting because of the more favorable antenna diagram. If this is not su cient, mechanical tilting is employed. Again, as with the azimuth these heuristics give us a sensible starting point, which allows us to restrict some local search heuristic to nearby settings.



(a) Berlin*

(b) Lisbon

Figure . : Azimuth optimization

5.4.3 Snapshots

Simulations are required, to realistically evaluate an scenario. This leads to the question whether it is possible to combine simulation and optimization. To this end, we will first try to model the core of a simulator as a .

Recalling the notation from Section . . the network to evaluate is given as a set of installations $\mathcal{I}$. We introduce binary variables x_{mi} which indicate whether mobile $m \in \mathcal{M}$ is served by installation $i \in \mathcal{I}$. It is required that no mobile is served by more than one installation:

$$\sum_{i \in \mathcal{I}} x_{mi} \leqslant 1 \quad \text{for all } m \in \mathcal{M} \qquad (.)$$

Adding inequality (.) we obtain for the uplink:

$$\frac{\gamma^{\uparrow}_{mi} p^{\uparrow}_{m}}{\bar{p}^{\uparrow}_{i} - \gamma^{\uparrow}_{mi} \alpha^{\uparrow}_{m} p^{\uparrow}_{m}} \geqslant \mu^{\uparrow}_{m} x_{mi} \quad \text{for all } m \in \mathcal{M}, i \in \mathcal{I}$$

And using inequality (.) for the downlink we obtain:

$$\frac{\gamma^{\downarrow}_{im} p^{\downarrow}_{im}}{\gamma^{\downarrow}_{im} \bar{\omega}_{im} \left(\bar{p}^{\downarrow}_{i} - \alpha^{\downarrow}_{m} p^{\downarrow}_{im}\right) + \sum_{j \neq i} \gamma^{\downarrow}_{jm} \bar{p}^{\downarrow}_{j} + \eta_{m}} \geqslant \mu^{\downarrow}_{m} x_{mi} \quad \text{for all } m \in \mathcal{M}, i \in \mathcal{I} \qquad (.)$$

Linearization

Since the above inequalities are quadratic, we have to find a linear reformulation for them, since the solution of quadratic mixed integer programming problems is di cult in practice. We show the linearization for the downlink case, the uplink is similar. Starting with a transformation of inequality (.):

$$\frac{\gamma^{\downarrow}_{im} p^{\downarrow}_{im}}{\mu^{\downarrow}_{m}} \geqslant \underbrace{\Big(\gamma^{\downarrow}_{im} \bar{\omega}_{im} \bar{p}^{\downarrow}_{i} - \gamma^{\downarrow}_{im} \bar{\omega}_{im} \alpha^{\downarrow}_{m} p^{\downarrow}_{im} + \sum_{j \neq i} \gamma^{\downarrow}_{jm} \bar{p}^{\downarrow}_{j} + \eta_m\Big)}_{\text{Total interference } \phi(i,\,m)} x_{mi} \geqslant 0$$

we find an upper bound for $\phi(i, m)$ by inserting the maximum total output power at each installation:

$$\Theta_{im} := \gamma^{\downarrow}_{im} \bar{\omega}_{im} \Pi^{\max\downarrow}_{i} + \sum_{j \neq i} \gamma^{\downarrow}_{jm} \Pi^{\max\downarrow}_{j} + \eta_m$$

Writing

$$\frac{\gamma^{\downarrow}_{im}}{\mu^{\downarrow}_{m}} p^{\downarrow}_{im} \geqslant \phi(m, i) - \Theta_{im}(1 - x_{mi}) \qquad (\ . \ )$$

we obtain the linearization of (.). Inequality (.) is always fulfilled if $x_{mi} = 0$. In case $x_{mi} = 1$ it is fulfilled if and only if inequality (.) is fulfilled.

One disadvantage of this approach is that we have modeled the as the di erence between signal and interference and not as the ratio between them. Some services such as file-transfer are packet-switched instead of the traditional circuit-switched. This allows to *upgrade* or *downgrade* their data rate and, correspondingly, their target depending on the utilization of the network. With our "basic" approach used here, we have to decide in advance which data rate we choose.

Taking the inequalities for uplink, downlink, and pilot together with limits on the power output and maximizing the number of served mobiles gives us essentially a snapshot evaluator. Experiments have shown that the linear relaxation of this model is quite tight.[23] Some decisions from the *Radio-Resource-Management* ( ), like preferring voice users over streaming users, can be incorporated by a suitable objective function. Furthermore, we are able to require minimum power levels as resulting from perfect power control by adding the power variables with some tiny costs to the objective function. Figure . . shows the result of an evaluation. Blue dots indicate served mobiles, red dots represent unserved mobiles. The gray area is the *region of interest* and a darker gray indicates higher terrain.

Optimization

To extend this model to optimize the network, we need only two additions: First, we extend the set of installations to include all potential installations and require that only a limited number of installations is chosen per site. Second, we have to change the objective to minimize the cost of the resulting network while still serving most of the users. We denote by $\mathcal{S}$ the set of all possible sites and introduce binary variables s_k, $k \in \mathcal{S}$ with the interpretation $s_k = 1$ if and only if site k is used. This allows us to

23 In a way, the relaxation resembles soft-handover since mobiles can be served by more than one cell.

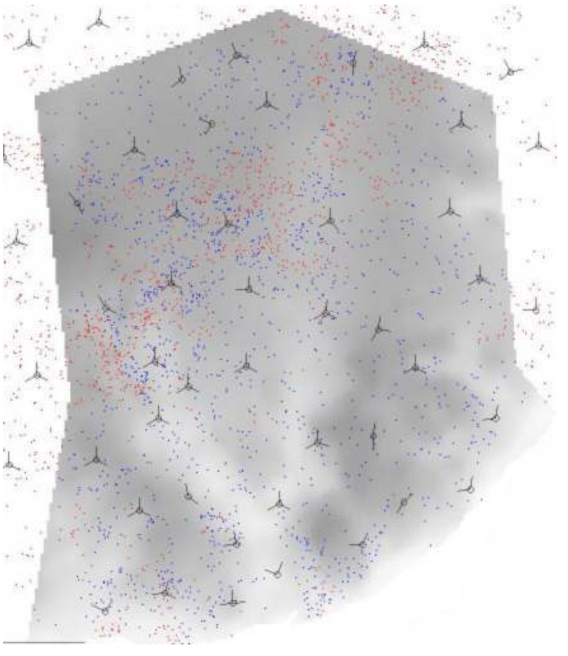


Figure . : Snapshot evaluation for Lisbon

introduce costs for opening a site[24]. Installations are selected through binary variables z_i, $i \in \mathcal{I}$, where $z_i = 1$ if and only if installation i is used. An installation i is only available if its site $\sigma(i)$ is in use:

$$z_i \leqslant s_{\sigma(i)} \quad \text{for all } i \in \mathcal{I} \qquad (.)$$

Of course, the number of installations at a site is bounded:

$$\Upsilon_k^{\min} \leqslant \sum_{i \in \mathcal{I}(k)} z_i \leqslant \Upsilon_k^{\max} \quad \text{for all } k \in \mathcal{S} \qquad (.)$$

and only a selected installation may serve mobiles:

$$x_{mi} \leqslant z_i \quad \text{for all } m \in \mathcal{M}, i \in \mathcal{I} \qquad (.)$$

Up to now we maximized the number of served mobiles, but with costs for opening sites and selecting installations. We would like to change the objective function to minimize network cost while still serving "enough" users. Operators, asked how many users are enough, usually state that: "*it's OK if, say, 5% of the mobiles are not served.*" Opening a site only to serve a "few" additional users is obviously a bad idea in this setting. A possible solution is to change inequality (.) to:

$$\sum_{i \in \mathcal{I}} x_{mi} = 1 - u_m \quad \text{for all } m \in \mathcal{M} \qquad (.)$$

24 Opening a site is a costly decision. It is, in fact, so expensive compared to installing another antenna at a site, that most sites are built with three antennas from the beginning on.

Either the mobile is served, or a binary variable u_m is set to one, indicating that mobile m is unserved. Adding

$$\sum_{m \in \mathcal{M}_d} u_m \leqslant 0.05\,|\mathcal{M}_d| \quad \text{for all } d \in \mathcal{D}$$

would request that in each snapshot at most % of the mobiles remain unserved. In computational experiments we used an alternative approach. The u_m variables were given high costs which would then trigger the opening of additional sites. This allows to eventually express how desirable it is to serve a mobile. The experiments have shown that the above model is computationally very unfavorable. This has several reasons:

- Monte-Carlo simulators use thousands of snapshots to get reliable results. To do the same, we have to include all the snapshots into a single .
- The grows by $\mathcal{I} \times \mathcal{M}$. The growth resulting from increasing the number of mobiles can partly be confined by preprocessing, since the number of installations that can possibly be the best server for a mobile is more or less constant.
- If a large number of possible installations per site is used, these will inevitably be rather similar. The more so if only average predictions like - are used.
- The linearization is a so-called *big M* formulation. These tend to be numerically unfavorable.
- The high cost di erence between serving a mobile compared to not serving it leads in case of a not serviceable mobile to an unfortunate branching order in the codes.

Apart from this, a uniform distribution of the unserved mobiles is desirable. In dense urban areas, which are our main focus, capacity is often the bottleneck and not coverage. It is quite possible that given the freedom to choose, aggregated dropping might take place in hot-spot areas.

For these reasons, we tried equation (.), taking the view that if it does not matter whether a mobile at a specific place is served, none should be generated at this place in a snapshot, anyway.

$$\sum_{i \in \mathcal{I}} x_{mi} = 1 \quad \text{for all } m \in \mathcal{M} \qquad (.)$$

This improved the computational solvability, but the range of feasibility for a given set of parameters became very small, e. g., all parameters had to be chosen extremely carefully to get feasible solutions. This gets increasingly di cult with every additional snapshot included in the problem.

In the end, we were not able to solve a suitable number of snapshots, i. e., more than one to get reliable results on realistic scenarios, even when we restricted the model to only the downlink case[25].

25 This is not that big a restriction, because if a mobile is not served, it will be either because of uplink, downlink or pilot requirements. This means two of the three inequalities are likely to be non-tight in a

5.5 Practice

We conclude our examination with an example. As we have seen is a complex system, where nearly everything is interacting with everything else. The highly nonlinear behavior of the system where problems in one area are proliferated to neighboring cells via interference, make it hard to correctly evaluate the performance of a network. Furthermore there is no clear definition of a "good" network. There are only several performance indicators that have to be weighted against each other.

We have seen limited evaluations of di erent scenarios with the M dynamic and advanced static simulators, with Forsk's A , the A T simulator and /Atesio's N V swift performance analyzer. While the qualitative results were mostly comparable, i. e., all tools see a high load in areas with a lot of traffic, the quantitative results about how many users a network can accommodate are very dependent on the actual settings of numerous parameters.

Until we have the possibility to compare results with the real world, we can only demonstrate our ability to model the problem, adapt to the evaluation tool at hand and produce sensible results using an ever increasing toolbox of models and methods. But then, this is what rapid mathematical programming is all about.

This said, we will now show the results of computing a network for the M Berlin public scenario using the models presented in the previous sections. The result will be compared with the initial network for the scenario, which was provided by e·plus and which is derived from their network.

The scenario itself has an area of square km. We defined a border strip of m that we excluded from the evaluation, leaving about square km e ective. In the main business hour on average users are active at all times, half of them using voice telephony and the rest distributed on the other services.

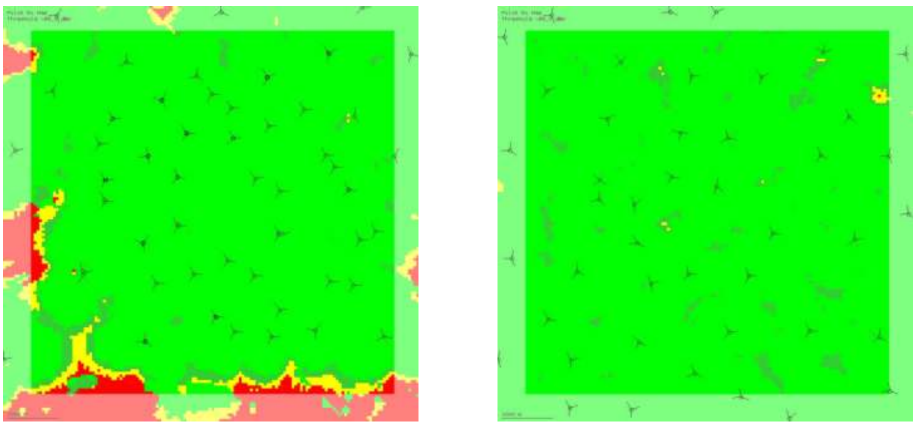
To construct a network, we first used the set covering model of Section . . , selecting from potential sites. Next the azimuth of the antennas was determined with the graph based model according to Section . . . Afterwards the tilts were set analytically. Finally, a local search heuristic was run, which repeatedly tried for individual promising sites to vary the tilt or change the azimuth by ± ° until no further improvement could be achieved. The original network and the one we computed were evaluated with the N V swift performance analyzer. The results can be seen in Table . . Even though we used cells fewer than the original network in the focus area, the performance of the computed network is better for the assumed tra c.

Figure . shows a comparison of the pilot coverage areas.[26] Note the light colored border which is not included in the evaluation. Red and yellow indicate a weak pilot signal. Areas with insu cient pilot coverage are unlikely to receive any service. This is independent of the load of the network.

Figure . visualizes the di erence in signal strength between the strongest and sec-

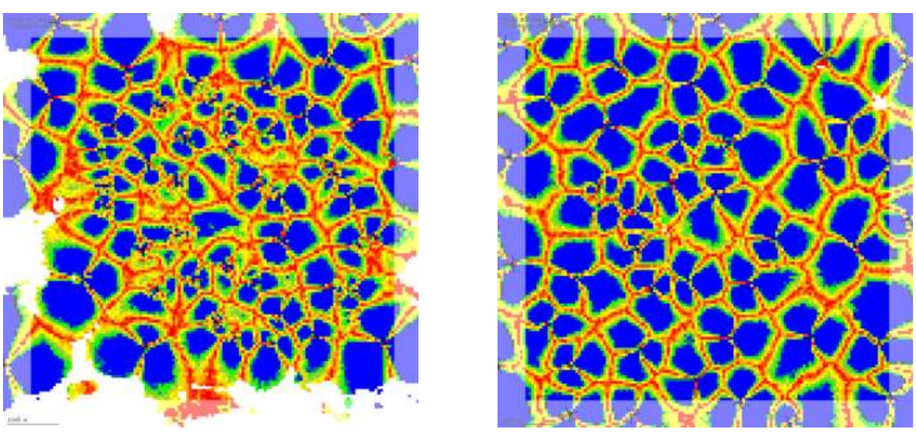
solution anyway. And since we are mostly in a downlink limited scenario, the tight one will be usually the downlink constraint.

26 This is one of the more reliable indicators, as it is only depending on the pathloss predictions.



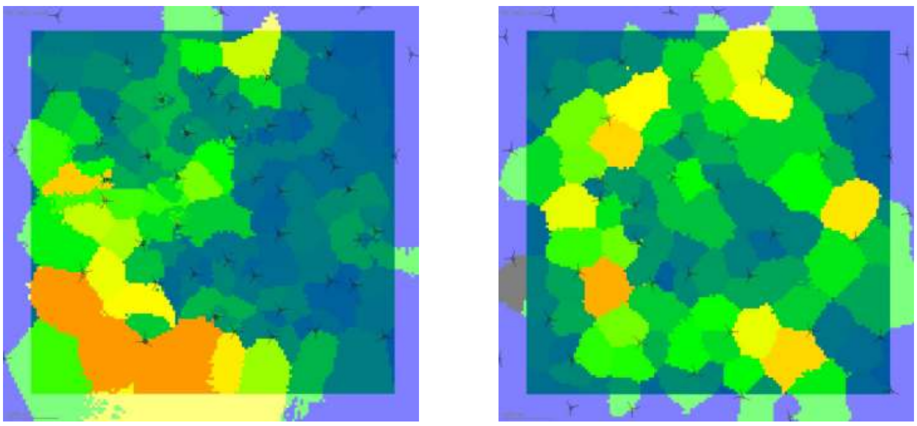
(a) Original (b) Computed

Figure . : Pilot coverage



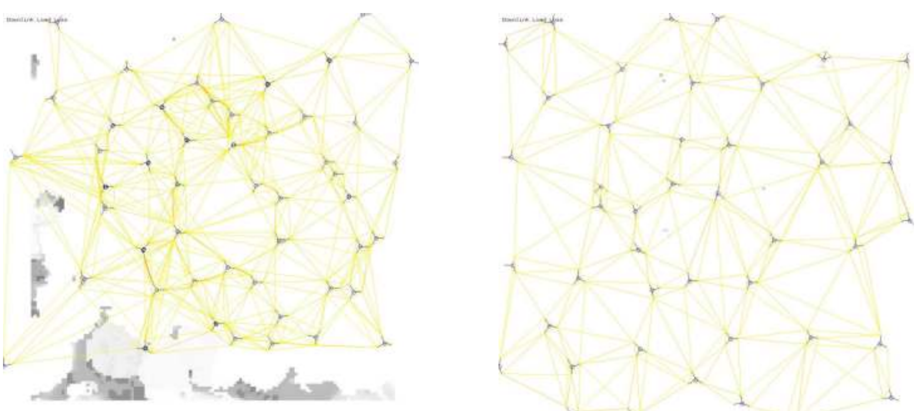
(a) Original (b) Computed

Figure . : Pilot separation



(a) Original (b) Computed

Figure . : Downlink cell load



(a) Original (b) Computed

Figure . : Load loss / interference coupling

	Original	Computed
Sites (inside focus zone)	50 (44)	44 (30)
Cells (inside focus zone)	148 (131)	132 (90)
Lower values indicate better performance		
Weak pilot signal % of area	6	1
Pilot pollution % of area	9	3
Uplink load (average) %	12	13
(maximum) %	40	36
Downlink load (average) %	20	20
(maximum) %	70	65
Total emitted power dBm	582	534
Overloaded cells	3	0
Lost traffic %	8	$\leqslant$ 1

Table . : Comparison of performance indicators

ond strongest pilot signal. A big di erence indicates that the cell with the strongest signal is "undisputed". Blue areas have at least dB di erence, green is down to dB, yellow starts at dB, and is getting more reddish up to zero decibel, which is colored plain red.[27] Pilot pollution happens if more than three pilot signals are within dB of each other. This is an indication that too many antennas are covering a specific area and unnecessary interference is likely to occur.

The distribution of the downlink load is shown in Figure . . Dark colors indicate a low load, lighter colors indicate higher load. Table . shows that while the average load of both networks is similar, the maximum load of the original network is higher. Note that % is the maximal possible downlink load in our case. Cells exceeding this threshold are overloaded and tra c will be lost. Additionally, overloaded cells cannot be expected to accommodate new users, i. e., users in the orange areas of the original network (Figure . a) initiating a call or moving into the area with an active connection are likely to be rejected.

Finally, Figure . shows two unrelated issues. The gray areas indicate load loss. As we can see, most of the loss in the original network is located in the lower left corner. Our investigations revealed that the existing sites have no possibility to cover that area completely. If five additional sites in the border area would be present as in the computed scenario, we could expect that the weak pilot and load loss indicators reach similar levels as in the computed network.

The graph imposed on the load loss map visualizes the interference coupling between the cells. Cells that do not have a connecting arc have no noteworthy mutual interference. Note that the arcs indicate how strong the coupling between the cells is, and not how strong the actual interference is. Looking at Figures . a we see a few or-

27 Areas with less than dB di erence are likely to be soft-handover areas, which is not bad per se.

ange arcs indicating that the strongest interference coupling happens if two antennas are directed at each other. Comparing with Figures . b and . a suggests that the results of our azimuth selection model are indeed beneficial.

The importance of less interference coupling can be seen if we increase the tra c demands. For a test we uniformly doubled the tra c in the scenario. Despite having considerably fewer cells, i. e., potential capacity, our new network performed comparably[28] to the original one.

5.5.1 Conclusion

planning is still in its infancy. We presented models that seem to work acceptably, but lack feedback from the real world to verify data, models, and results. Once the models are more mature, additional theoretical work is needed to get a better notion of the potential capacity of a network. In contrast to networks, the introduction of an additional site can decrease the capacity of an network. At the moment, it is very di cult to give more than a trivial lower bound on the number of cells needed to handle a given tra c load. In face of the uncertainty regarding the underlying data, especially the pathloss predictions, all results about networks should be viewed with some suspicion.

Nevertheless, rapid mathematical prototyping has proved itself a formidable tool to quickly investigate lines of thought and to di erentiate between promising and futile approaches. Additional details, further information on models, data, and advanced topics can be found in Eisenblätter et al. (a,b,c,d,e,f,), Amaldi et al. (, a,b), Mathar and Schmeink ( ), Whitaker and Hurley ( ).

5.5.2 Acknowledgements

Some of the graphics shown in this chapter were computed using data provided by e·plus Mobilfunk GmbH & Co. KG, Düsseldorf, as part of the M project. This data in turn was generated using data provided from the Bundesamt für Kartographie und Geodäsie, Frankfurt am Main, and from Tele Atlas N.V., Düsseldorf.

28 Both networks had most cells overloaded and lost % to % of the tra c.

Chapter 6

Steiner Tree Packing Revisited

And now something completely different
— BBC

In this chapter we will explore how far we can get with our rapid prototyping approach on a "classical" hard combinatorial problem. Bob Bixby (Bixby et al., 2000, Bixby, 2002) claims "dramatic" improvements in the performance of generic -Solvers like . We will revisit the Steiner tree packing problem in graphs and make some comparisons against special purpose codes. I wish to thank Alexander Martin and David Grove Jørgensen for their help and contributions.

6.1 Introduction

The weighted Steiner tree problem in graphs ( ) can be stated as follows:

> *Given a weighted graph* $G = (V, E, c)$ *and a non-empty set of vertices* $T \subseteq V$ *called* terminals, *find an edge set* S^* *such that* $(V(S^*), S^*)$ *is a tree with minimal weight that spans* T.

This problem is nearly as classical as the *Traveling Salesman Problem* ( ). An extensive survey on the state-of-the-art of modeling and solving the can be found in Polzin ( ).

Most papers on the claim[1] real-world applications, especially in -design and wire-routing. This usually refers to a generalization of the , the weighted Steiner tree packing problem in graphs ( ). Instead of having one set of terminals, we have N non-empty disjoint sets $T_1, \ldots, T_N$, called *Nets*, that have to be "packed" into the graph simultaneously, i. e., the resulting edge sets $S_1, \ldots, S_N$ have to be disjoint. In these applications, G is usually some kind of grid graph.

1 Chvátal ( ) describes solving the as sports. The qualifies as sports for similar reasons.

Grötschel et al. ( ), Lengauer ( ) give detailed explanations of the modeling requirements in -design. We will follow their classification and give an overview of the main variants only.

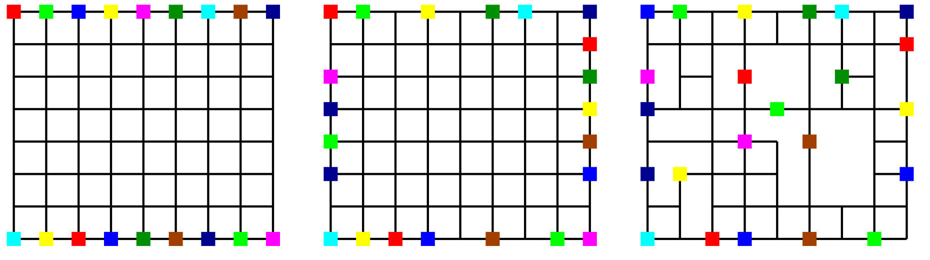
Routing

Motivated by the applications three routing models are of particular interest:

Channel routing (. a) Here, we are given a complete rectangular grid graph. The terminals of the nets are exclusively located on the lower and upper border. It is possible to vary the height of the channel. Hence, the size of the routing area is not fixed in advance. Usually all nets have only two terminals, i. e., $|T_i| = 2$.

Switchbox routing (. b) Again, we are given a complete rectangular grid graph. The terminals may be located on all four sides of the graph. Thus, the size of the routing area is fixed.

General routing (. c) In this case, an arbitrary grid graph is considered. The terminals can be located arbitrarily (usually at some hole in the grid).



(a) Channel routing (b) Switchbox routing (c) General routing

Figure . : routing variations

Intersection model

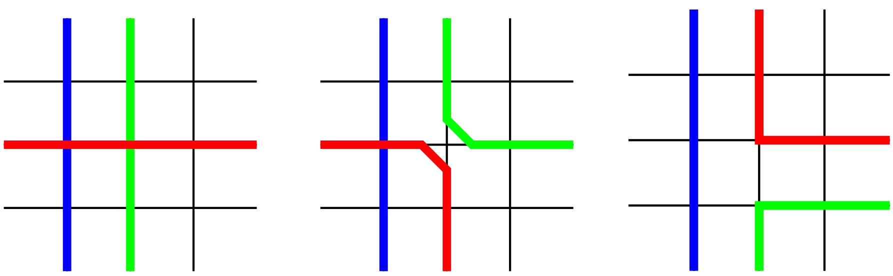
The intersection of the nets is an important point in Steiner tree packing. Again three di erent models are possible.

Manhattan (. a) Consider some (planar) grid graph. The nets must be routed in an edge disjoint fashion with the additional restriction that nets that meet at some node are not allowed to bend at this node, i. e., so-called *Knock-knees* are not allowed. This restriction guarantees that the resulting routing can be laid out on two layers at the possible expense of causing long detours.

Knock-knee (. b) Again, some (planar) grid graph is given and the task is to find an edge disjoint routing of the nets. In this model Knock-knees are possible.

Very frequently, the wiring length of a solution in this case is smaller than in the Manhattan model. The main drawback is that the assignment to layers is neglected.

Node disjoint (. c) The nets have to be routed in a node disjoint fashion. Since no crossing of nets is possible in a planar grid graph, this requires a multi-layer model.



(a) Manhattan model (b) Knock-knee model (c) Node disjoint model

Figure . : intersection models

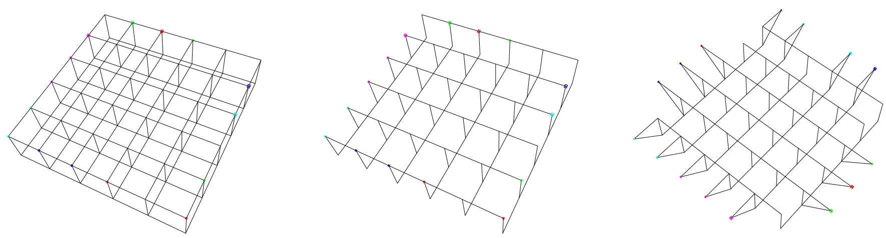
Multiple layers

While channel routing usually involves only a single layer, switchbox and general routing problems are typically multi-layer problems. Using the Manhattan and Knock-knee intersection is a way to reduce the problems to a single-layer model. Accordingly, the multi-layer models typically use node disjoint intersection. While the multi-layer model is well suited to reflect reality, the resulting graphs are in general quite large. We consider two[2] possibilities to model multiple layers:

k-**crossed layers** (. a) There is given a k-dimensional grid graph (that is a graph obtained by stacking k copies of a grid graph on top of each other and connecting corresponding nodes by perpendicular lines, so-called *vias*), where k denotes the number of layers. This is called the k-layer model in Lengauer ( ).

k-**aligned layers** (. b) This model is similar to the crossed-layer model, but in each layer there are only connections in one direction, either east-to-west or north-to-south. Lengauer ( ) calls this the *directional* multi-layer model. Korte et al. ( ) indicate that for $k = 2$ this model resembles the technology used in -wiring best. Boit ( ) mentions that current technology can use a much higher number of layers.

2 A third possibility is to use a single-layer model with edge capacities greater than one.



(a) Multi-crossed layers (b) Multi-aligned layers (c) With connectors

Figure . : modeling taxonomy

Note that for switchbox routing there is a one-to-one mapping between feasible solutions for the Manhattan one-layer model ( ) and the node disjoint two-aligned-layer model ( ), assuming that there are never two terminals on top of each other, i. e., connected by a via.

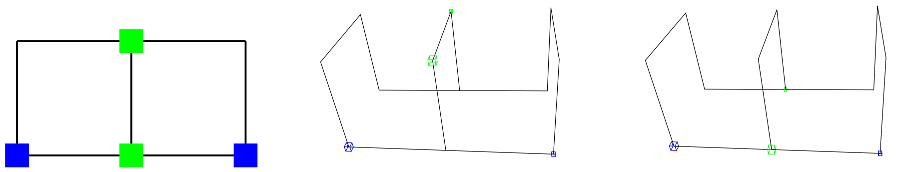
To map a feasible solution for to , all we have to do is to merge the two layers, i. e., contract every pair of nodes along the respective via. Since no two terminals are connected by a via, the situation that two terminals are to be merged cannot happen. Due to the shape of the aligned-layer-graph, the result obviously adheres to the Manhattan constraint and is edge disjoint as no edges were added. Finally, because all nodes that were connected by vias in the solution are now contracted, all paths between terminals are still present, accordingly the new solution has to be feasible.

In the other direction (to), we assign vias as needed, this is straightforward. The result will be node disjoint, because whenever two nets cross in the solution they are now assigned to di erent layers and all other nodes in the graph are touched by at most one net. More di cult is to decide the layer for each terminal. Have a look at Figure . for an example. If a terminal is located at a corner of the grid graph it can be assigned to either node, because it will block the whole corner anyway. In case a terminal is not located at a corner (Figure . a), it has to be placed in the same layer as the edge perpendicular to the border (Figure . b), because otherwise it might block another net (Figure . c).

For the general routing model, the above transformation might not be possible. If a terminal is within the grid there is no easy way to decide the correct layer for the terminal in the two-layer model.

Unfortunately, in the seven "classic" instances given by Burstein and Pelavin ( ), Luk ( ), Coohoon and Heck ( ) two terminals are connected to a single corner in several cases. This stems from the use of *connectors*, i. e., the terminal is outside the grid and connected to it by a dedicated edge. In the multi-layer models there has to be an edge from the terminal to all permissible layers (Figure . c).

The Knock-knee one-layer model can also be seen as an attempt to approximate the node disjoint two-crossed-layer model. But mapping between these two models



(a) Manhattan one-layer (b) Feasible (c) Infeasible

Figure . : Manhattan one-layer vs. Node disjoint two-aligned-layer

is not as easy. Brady and Brown ( ) have designed an algorithm that guarantees that any solution in the Knock-knee one-layer model can be routed in a node disjoint four-crossed-layer model, but deciding whether three layers are enough is shown to be $\mathcal{NP}$-complete by Lipski ( ).

For an example have a look at Figure . . Figures . a and . d show two feasible Knock-knee one-layer routings. If we solve the same problem in the node disjoint crossed-multi-layer model, the first example needs at least two layers (Figure . b), while the second needs at least three layers (Figure . e). But this holds only if we can choose the layers for the terminals. If the layers are fixed in advance in both cases up to four layers may be needed (Figures . c and . f).

6.2 Integer programming models

A survey of di erent integer programming models for Steiner tree packing can be found in Chopra ( ). We will examine two of the models in more detail.

6.2.1 Undirected partitioning formulation

This formulation is used in Grötschel et al. ( ). Given a weighted grid graph $G = (V, E, c)$, and terminal sets $T_1, \ldots, T_N$, $N > 0$, $\mathcal{N} = \{1, \ldots, N\}$, we introduce binary variables x^n_{ij} for all $n \in \mathcal{N}$ and $(i,j) \in E$, where $x^n_{ij} = 1$ if and only if edge $(i,j) \in S_n$. We define $\delta(W) = \{(i,j) \in E | (i \in W, j \notin W) \vee (i \notin W, j \in W)\}$ with $W \subseteq V$.

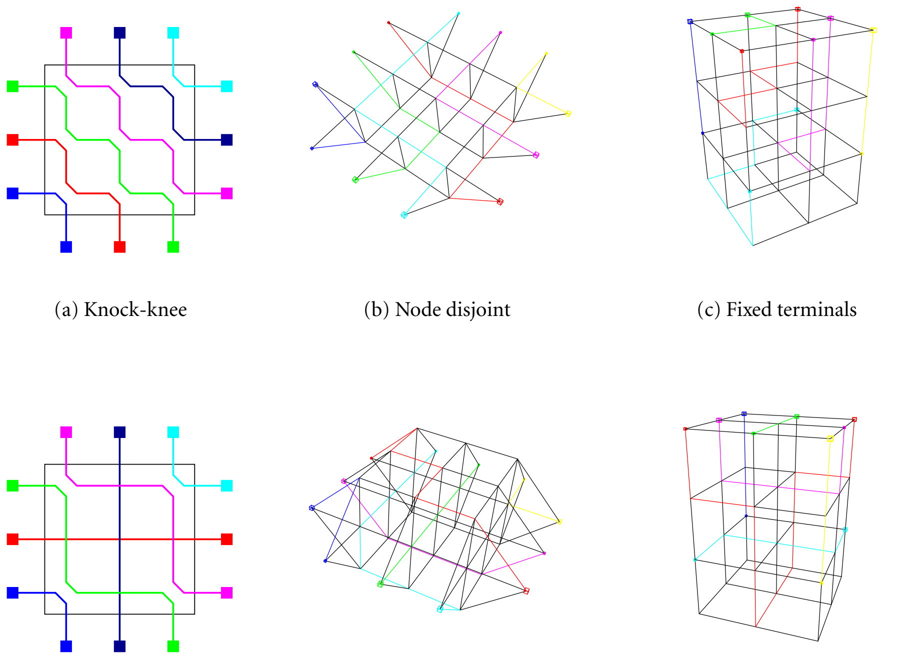
The following formulation models all routing choices for the Knock-knee one-layer model:

$$\min \quad \sum_{n\in\mathcal{N}} \sum_{(i,j)\in E} c_{ij} x^n_{ij}$$

$$\sum_{(i,j)\in\delta(W)} x^n_{ij} \geqslant 1 \qquad \text{for all } W \subset V, W \cap T_n \neq \emptyset, (V \setminus W) \cap T_n \neq \emptyset, n \in \mathcal{N} \qquad (\ .\ )$$

$$\sum_{n\in\mathcal{N}} x^n_{ij} \leqslant 1 \qquad \text{for all } (i,j) \in E \qquad (\ .\ )$$

$$x^n_{ij} \in \{0,1\} \qquad \text{for all } n \in \mathcal{N}, (i,j) \in E \qquad (\ .\ )$$



(a) Knock-knee (b) Node disjoint (c) Fixed terminals

(d) Knock-knee (e) Node disjoint (f) Fixed terminals

Figure . : Number of layers needed to route a Knock-knee one-layer solution

In order to use Manhattan intersection another constraint is needed to prohibit Knock-knees. Let (i, j), (j, k) be two consecutive horizontal (or vertical) edges. Then,

$$\sum_{n \in N_1} x_{ij}^n + \sum_{m \in N_2} x_{jk}^m \leqslant 1 \text{for all } j \in V, N_1 \subset \mathcal{N}, N_2 \subset \mathcal{N}, N_1 \cap N_2 = \emptyset, N_1 \cup N_2 = \mathcal{N} \quad (.)$$

is called *Manhattan inequality*.[3] The model can be further strengthened with several valid inequalities as described in Grötschel et al. (a,b), Grötschel et al. ( ).

6.2.2 Multicommodity flow formulation

For our computational investigations we will use a multicommodity flow formulation. For the this was formulated by Wong ( ). The multicommodity flow formulation has the advantage that it has only a polynomial number of variables and constraints.

3 Another way to enforce the Manhattan constraints is given in Jørgensen and Meyling ( ). Every node v in the grid graph is split into two nodes v_h and v_v. v_h is connected to the horizontal edges and v_v to the vertical edges incident to v. An additional edge is used to connect v_h and v_v. This makes it impossible for more than one net to use both vertical and horizontal edges incident to v. Note, that this is equivalent to converting an one-layer model into a two-aligned-layer model.

This allows us to generate the complete model in advance and solve it with a standard solver like .

Given a weighted bidirectional grid digraph $G = (V, A, c)$, and sets $T_1, \dots, T_N$, $N > 0$, $\mathcal{N} = \{1, \dots, N\}$ of terminals, we arbitrarily choose a root $r_n \in T_n$ for each $n \in \mathcal{N}$. Let $R = \{r_n | n \in \mathcal{N}\}$ be the set of all roots and $T = \bigcup_{n \in \mathcal{N}} T_n$ be the union of all terminals. We introduce binary variables $\bar{x}^n_{ij}$ for all $n \in \mathcal{N}$ and $(i, j) \in A$, where $\bar{x}^n_{ij} = 1$ if and only if arc $(i, j) \in S_n$. Additionally we introduce non-negative variables y^t_{ij}, for all $t \in T \setminus R$. For all $i \in V$, we define $\delta^+_i := \{(i, j) \in A\}$ and $\delta^-_i := \{(j, i) \in A\}$. For all $t \in T_n$, $n \in \mathcal{N}$, we define $\sigma(t) := n$. The following formulation models all routing choices for any number of layers, crossed and aligned, with Knock-knee intersection:

$$\min \sum_{n \in \mathcal{N}} \sum_{(i,j) \in A} c^n_{ij} \bar{x}^n_{ij}$$

$$\sum_{(i,j) \in \delta^-_j} y^t_{ij} - \sum_{(j,k) \in \delta^+_j} y^t_{jk} = \left\{ \begin{array}{rl} 1 & \text{if } j = t \\ -1 & \text{if } j = r_{\sigma(t)} \\ 0 & \text{otherwise} \end{array} \right\} \quad \text{for all } j \in V, t \in T \setminus R \qquad (\ .\ )$$

$$0 \leqslant y^t_{ij} \leqslant \bar{x}^{\sigma(t)}_{ij} \quad \text{for all } (i, j) \in A, t \in T \setminus R \qquad (\ .\ )$$

$$\sum_{n \in \mathcal{N}} (\bar{x}^n_{ij} + \bar{x}^n_{ji}) \leqslant 1 \quad \text{for all } (i, j) \in A \qquad (\ .\ )$$

$$\bar{x}^n_{ij} \in \{0, 1\} \quad \text{for all } n \in \mathcal{N}, (i, j) \in A \qquad (\ .\ )$$

To use node disjoint intersection we have to add:

$$\sum_{n \in \mathcal{N}} \sum_{(i,j) \in \delta^-_j} \bar{x}^n_{ij} \leqslant \left\{ \begin{array}{ll} 0 & \text{if } j \in R \\ 1 & \text{otherwise} \end{array} \right. \quad \text{for all } j \in V \qquad (\ .\ )$$

The above system (especially (.)) has the following implications which hold also for the linear relaxation, i. e., $\bar{x}^n_{ij} \in [0, 1]$ instead of (.):

$$\bar{x}^n_{ij} \geqslant \max_{t \in T_n \setminus R} y^t_{ij} \quad \text{for all } (i, j) \in A, n \in \mathcal{N} \qquad (\ .\ )$$

Assuming $c^n_{ij} > 0$, inequality (.) is even met with equality. This leads to

$$\bar{x}^n_{jk} \leqslant \sum_{(i,j) \in \delta^-_j} \bar{x}^n_{ij} \quad \text{for all } j \in V \setminus R, (j, k) \in \delta^+_j, n \in \mathcal{N}\ , \qquad (\ .\ )$$

i. e., for each net the flow on each outgoing arc is less than or equal to the total flow into the node.

Proof. For each $t \in T \setminus R$ and each $j \in V \setminus T$ equation (.) states that $\sum_{(i,j) \in \delta^-_j} y^t_{ij} = \sum_{(j,k) \in \delta^+_j} y^t_{jk}$. It follows that $\sum_{(i,j) \in \delta^-_j} y^t_{ij} \geqslant y^t_{jk}$ for any $(j, k) \in \delta^+_j$ and further $\sum_{(i,j) \in \delta^-_j} \max_{t \in T_n \setminus R} y^t_{ij} \geqslant \max_{t \in T_n \setminus R} y^t_{jk}$ for any $(j, k) \in \delta^+_j$. Substituting (.) we arrive at (.). This holds also if $j \in T \setminus R$, because (.) only limits the flow on outgoing arcs in this case. □

It is possible to strengthen (.) by subtracting the incoming arc anti-parallel to the outgoing arc in question, giving the following valid inequality:

$$\bar{x}^n_{jk} + \bar{x}^n_{kj} \leqslant \sum_{(i,j)\in\delta^-_j} \bar{x}^n_{ij} \quad \text{for all } j \in V \setminus R, (j,k) \in \delta^+_j, n \in \mathcal{N} \ , \qquad (\ . \ )$$

Proof. If in any optimal solution $\bar{x}^n_{kj}$ is one, $\bar{x}^n_{jk}$ has to be zero due to (.). In this case (.) is trivially satisfied. In case $\bar{x}^n_{kj}$ is zero, (.) is equal to (.). □

6.2.3 Comparison of formulations

Theorem 2. *Any feasible solution of the relaxation of the multicommodity flow formulation of the node disjoint two-aligned-layer model together with inequality (.) defines a feasible solution for the relaxation of the partitioning formulation of the Manhattan one-layer model by setting* $x^n_{ij} = \bar{x}^n_{ij} + \bar{x}^n_{ji}$ for all $(i,j) \in E$.

Proof. For any given $n \in \mathcal{N}$ and any given partition $W \subset V$, $W \cap T_n \neq \emptyset$, and $U = (V \setminus W) \cap T_n \neq \emptyset$, we can assume without loss of generality that $r_n \in R \cap U$ and that there exists a terminal $t \in (T_n \setminus R) \cap W$. Due to (.) $\{(i,j) \in A \mid y^t_{ij}\}$ form a path from r to t, i. e., any feasible solution to (.) will constitute a flow of one unit within the y^t variables from r_n to t. It follows that the sum of the y^t in the cut between U and W is at least one. Due to (.) the same holds for the $\bar{x}^n$, i. e., $\sum_{(i,j)\in A, i\in U, j\in W} \bar{x}^n_{ij} \geqslant 1$. Consequently (.) holds. (.) and (.) hold because of (.) and (.).

In the two-aligned-layer model, each node j in the graph has at most three neighbors i, k, and l, with l being the node on the other layer.

Due to (.) for (.) to hold it su ces to show that $x^n_{ij} + x^m_{jk} \leqslant 1$ holds for any $j \in V$ and $m \neq n$. For the two-aligned-layer model we can rewrite this as:

$$x^n_{ij} + x^m_{jk} = \bar{x}^n_{ij} + \bar{x}^n_{ji} + \bar{x}^m_{jk} + \bar{x}^m_{kj} \leqslant 1 \qquad (\ . \ )$$

(i) For $j \in V \setminus T_n$ this holds because adding up

$$\begin{aligned} \bar{x}^n_{ij} + \bar{x}^n_{kj} + \bar{x}^n_{lj} + \bar{x}^m_{ij} + \bar{x}^m_{kj} + \bar{x}^m_{lj} &\leqslant 1 \quad \text{holds due to (.)} \\ \bar{x}^n_{ji} - \bar{x}^n_{kj} - \bar{x}^n_{lj} &\leqslant 0 \quad \text{holds due to (.)} \\ \bar{x}^m_{jk} - \bar{x}^m_{ij} - \bar{x}^m_{lj} &\leqslant 0 \quad \text{holds due to (.)} \end{aligned}$$

results in (.).

(ii) In case $j \in R$, (.) ensures that $\bar{x}^n_{ij} + \bar{x}^m_{kj} = 0$ and (.) proliferates this to the corresponding y^t variables. It follows from (.) that all $y^t_{ji} = 0$ for $(j,i) \in \delta^+_j$ with $\sigma(t) \neq \sigma(j)$. Since the m and n are from two disjoint nets, at most one of $\bar{x}^n_{ji}$ and $\bar{x}^m_{jk}$ can be non-zero and (.) holds.

(iii) In case $j \in T \setminus R$, (.) requires $\sum_{(i,j)\in\delta_j^-} y_{ij}^j = 1$. Due to (.), (.) this forces $y_{ij}^t = 0$ for all $(i,j) \in \delta_j^-$ with $\sigma(t) \neq \sigma(j)$. It follows from (.) that $y_{ji}^t = 0$ for all $(j,i) \in \delta_j^+$ with $\sigma(t) \neq \sigma(j)$. Since m and m are from two disjoint nets (.) holds.

□

Corollary 1. *The relaxation of the multicommodity flow formulation of the node disjoint two-aligned-layer model is strictly stronger than the relaxation of the partitioning formulation of the Manhattan one-layer model.*

Proof. Theorem implies that for the it is at least as strong. Polzin ( ) shows that for the the relaxation of the multicommodity flow formulation is equivalent to the directed cut formulation, which in turn is strictly stronger than the undirected partitioning formulation. It follows, that this holds for the , since the is a special case. □

6.3 Valid inequalities

While the flow formulation is strictly stronger than the partitioning formulation alone, it can be further strengthened by valid inequalities. Interestingly, the flow formulation does not ensure

$$\sum_{(i,j)\in\delta_j^-} \bar{x}_{ij}^n \leqslant \sum_{(j,k)\in\delta_j^+} \bar{x}_{jk}^n \quad \text{for all } j \in V \setminus (T \setminus R), n \in \mathcal{N} \qquad (.)$$

as can be seen in Figure . (Koch and Martin, , Polzin,). Numbers indicate arc weights, $T_1 = \{r, s\}$, $T_2 = \{r, t\}$, and $R = \{r\}$. Each arc (i,j) in Figure . corresponds to $y_{ij}^t = 0.5$. The objective function value is . for the relaxation and for a feasible integer solution. Adding (.) strengthens the relaxation to provide an objective function value of .

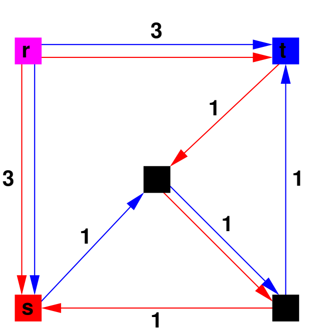


Figure . : The central node violates (.)

The critical cut inequalities introduced in Grötschel et al. (c) are also valid for the flow formulation. Consider a graph $G = (V, E)$ with unit edge capacities and a list of nets $\mathcal{N}$. For a node set $W \subseteq V$ we define $S(W) := \{n \in \mathcal{N} | T_n \cap W \neq \emptyset, T_n \cap (V \setminus W) \neq \emptyset\}$. The cut induced by W is called *critical* if $s(W) := |\delta(W)| - |S(W)| \leqslant 1$. The cut induced by W is critical if there is no edge disjoint path entering and leaving W left, i. e., no net without a terminal in W can pass through W in a feasible solution. In the node disjoint case the support of this cut can get even stronger, as can be seen for example in Figure . .

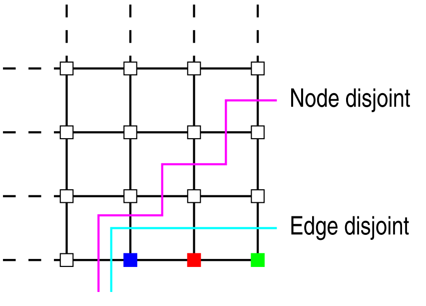
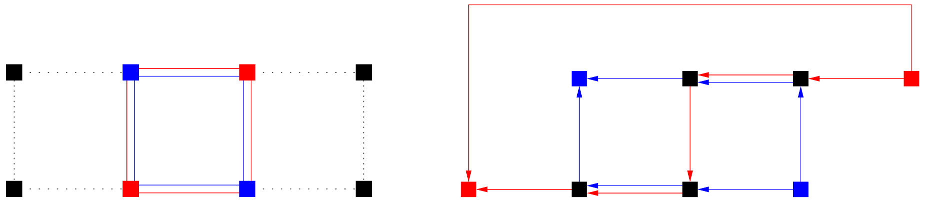


Figure . : Critical cut in the edge disjoint and node disjoint case

The grid inequality described in Grötschel et al. ( ) basically states that it is not possible for two nets T_1 and T_2 to cross each other in a $2 \times h$, $h \geqslant 2$ grid graph. As shown in Figure . a, there exist non integral solutions for the relaxation of the partitioning model in this case (unit edge weights, blue edges mean $x^1_{ij} = 0.5$, red edges indicate $x^2_{ij} = 0.5$). For an integral solution some path outside the $2 \times h$ grid is required. In the



(a) undirected Knock-nee

(b) directed node disjoint

Figure . : Grid inequalities

multicommodity flow formulation of the directed node disjoint model it is still possible to find a non-integral solution to the relaxation, even though the path outside the $2 \times h$ grid is already present. Figure . b shows the smallest example of this solution (unit arc weights, blue arcs correspond to $y^1_{ij} = 0.5$, red arcs indicate $y^2_{ij} = 0.5$). Note that at least a $3 \times h$ grid plus the two terminals of T_2 are needed for this configuration to occur.

6.4 Computational results

In this section we present computational results obtained by generating the complete integer program resulting from the directed multicommodity flow formulation with Z and then solving it with . . None of the strengthening inequalities and no preprocessing prior to was used to reduce the size of the problems. The versatility of our approach can be seen by the fact that we can generate general-routing problems, which includes channel and switchbox routing as a special case, with any number of layers, crossed and aligned, either with knock-knee or node disjoint intersection. The only missing case is Manhattan intersection, but this can always be reformulated as an equivalent multi-aligned-layer problem.

Table . lists all Steiner tree packing problem instances we have considered. N denotes the number of nets, and |T| the total number of terminals. The columns labeled *Vars*, *Cons*, and *NZ* list the number of variables, constraints and non-zero entries in the constraint matrix of the generated integer programs. Complete descriptions of all instances are listed in Appendix D. The Z program used for the node disjoint model can be found in Appendix C. on page .

We used three di erent sets of settings, denoted *(B)arrier*, *(D)efault* and *(E)mphasis*. The di erences are listed in Table . . The mixed integer optimality gap tolerance was set to zero in all settings.

Parameter	(B)arrier	(D)efault	(E)mphasis
Generate cuts	no	auto	no
Probing	yes	auto	yes
LP algorithm	barrier	dual simplex	dual simplex
MIP emphasis	balanced	balanced	feasibility

Table . : Comparison of settings used

6.4.1 Choosing the right LP solver algorithm

The barrier setting is quite uncommon, since usually the dual simplex is the preferred algorithm for solving the subproblems. The reason for this is the missing warm-start capability of the barrier or interior-point algorithm, which requires to solve each subproblem from scratch.[4] On the other hand, the running times for the barrier algorithm tend to be much more predictable and are sometimes nearly constant for each subproblem. Since we observed that the reoptimization of the subproblems using the dual simplex algorithm often took considerable more time and iterations than expected, we tried the barrier algorithm instead.[5]

4 For current developments in interior-point warm-start see, e. g., Gondzio and Grothey ( ), Elhedhli and Go n ( ).

5 The high number of simplex iterations needed to reoptimize a subproblem after branching on some variable may have the following reasons: Part of the pivots are degenerate. Additionally the basis has to be changed substantially to reach the optimum again. To understand the latter, remember what the model is

Table . shows the results for solving the root relaxation of *alue-4* (see Table .) with di erent algorithms. The times are wall clock times on a dual Alpha pro-cessor workstation with megahertz.[6] The first row of the table gives the time for the barrier algorithm without cross-over. The barrier algorithm is much better suited to parallelization than the simplex algorithm and accordingly we get a % speed-up by using the dual processor version as can be seen in the second row. The third row shows that performing a cross-over takes nearly twice the time of finding the solution in the first place. Next can be seen that on this selected instance the dual simplex algorithm needs more than eight times longer to solve the instance than the barrier algorithm.[7] In many studies (e. g. Grötschel et al., a) it is reported that perturbing the problem leads to significant speed-ups in the simplex algorithm. Since automatically applies perturbation after some time when solving the problem, we solved it again with explicitly switching perturbation on and supplying a very high perturbation constant of one.[8] Interestingly the results are disappointing. In the next two rows the experiment is repeated with the primal simplex algorithm with even worse results.[9]

Algorithm	Iterations	Time [s]
Barrier	17	357
Barrier (2 threads)	17	235
Barrier with cross-over	17	865
Dual simplex	112,789	3,032
Dual simplex with perturbation	192,831	4,930
Primal simplex	467,205	17,368
Primal simplex with perturbation	609,297	15,184

Table . : Solving the root relaxation of *alue-4*

As a consequence, we tried to use the barrier algorithm for the root relaxation and the dual simplex for the subproblems. Unfortunately this requires a cross-over from the non-vertex barrier solution to a vertex solution, which, as we have seen, takes considerable time. In the end we tried to use the barrier algorithm also for the subproblems. This caused a small problem with the cut generation in as the routine computing Gomory-cuts needs a vertex solution. Since as far as we have observed it, Gomory cuts are the only cuts applied to the problems, and since we are not sure about their impact, we disabled the cut generation completely for the *Barrier* and *Emphasis* settings.

about: Routing nets through a grid graph. Variables with non-integral values mean that at least two nets are competing for a route. By fixing such a variable at least one net has to be rerouted. Or, one net has a split route. In this case, due to the inherent symmetry in a grid graph, fixing a single variable will often result in a considerable rerouting of the net to reach another permutation of the former configuration.

6 A . gigahertz - is about . times faster for this task

7 automatically switches to devex pricing after a short time. Explicitly setting steepest edge pricing does neither change the running time nor the number of iterations needed significantly.

8 Although the objective function is all ones and zeros we know from experience that even with a highly perturbed objective function the result is likely to be near the optimum.

9 About half of the primal simplex iterations are needed to become feasible in phase .

Name	Size	N	|T|	Vars	Cons	NZ
Knock-knee one-layer model						
augmenteddense-1	16×18	19	59	70,918	62,561	215,158
dense-1	15×17	19	59	63,366	56,057	192,246
difficult-1	23×15	24	66	94,776	78,292	275,712
modifieddense-1	16×17	19	59	67,260	59,410	204,060
moredifficult-1	22×15	24	65	89,440	73,299	258,688
pedagogical-1	15×16	22	56	56,560	44,909	159,580
terminalintens-1	23×16	24	77	119,196	106,403	365,328
Node disjoint two-aligned-layer model						
augmenteddense-2	16×18	19	59	97,940	91,587	326,438
difficult-2	23×15	24	66	131,604	115,399	427,536
moredifficult-2	22×15	24	65	123,890	107,779	400,952
pedabox-2	15×16	22	56	77,168	65,067	245,576
terminalintens-2	23×16	24	77	164,010	154,947	550,104
sb11-20-7	21×21	7	77	197,274	243,726	751,884
sb3-30-26d	31×31	29	87	485,212	437,515	1,607,464
sb40-56	41×41	56	112	1,111,264	755,307	3,318,000
Node disjoint two-crossed-layer model						
gr2-8-32	9×9	8	32	22,144	21,038	76,512
Node disjoint three-aligned-layer model						
dense-3	15×17	19	59	144,668	131,412	482,722
modifieddense-3	16×17	19	59	154,580	140,307	515,986
taq-3	25×25	14	35	115,640	98,508	368,760
Node disjoint four-aligned-layer model						
alue-4	25×25	22	55	294,084	236,417	933,830

Table . : instances

6.4.2 Results for the Knock-knee one-layer model

Table . shows the results for the Knock-knee one-layer model. The column labeled *CS* contains the setting according to Table . . *B&B Nodes* denotes the number of Branch-and-Bound nodes including the root node evaluated by . *Root node* lists the objective function value of the relaxation of the root node. Finally *arcs* is the total number of arcs used in the optimal solution.

As we can see from the table, the relaxation is rather strong, but this is in line with other reported results like Martin ( ), Jørgensen and Meyling ( ). Since for *difficult-1, modifieddense-1, moredifficult-1,* and *pedabox-1* the relaxation already provides the optimal value, it is possibly to solve these instances without any branching. The Default setting achieves this in three of the four cases, which is exceptional as only general heuristics are used and in none of the cases the optimal solution of the root has

Name	CS	B&B Nodes	Time [s]	Root node	Arcs
augmenteddense-1	B	545	6,245	466.5	469
	D	53	7,277	466.5	469
	E	41	3,303	466.5	469
dense-1	B	189	1,954	438.0	441
	D	>300	>120,000	438.0	—
	E	64	15,891	438.0	441
difficult-1	B	1,845	17,150	464.0	464
	D	1	160	464.0	464
	E	15	274	464.0	464
modifieddense-1	B	33	358	452.0	452
	D	1	150	452.0	452
	E	3	132	452.0	452
moredifficult-1	B	489	4,102	452.0	452
	D	121	6,635	452.0	452
	E	6	118	452.0	452
pedabox-1	B	45	187	331.0	331
	D	1	35	331.0	331
	E	31	166	331.0	331
terminalintens-1	B	>7,000	>120,400	535.0	(536)
	D	15	2,779	535.0	536
	E	160	3,903	535.0	536

Table . : Results for the Knock-knee-one-layer model

been a feasible solution of the integer program.[10] The reason for this success may lie in the application of Gomory cuts which is only done in the Default settings.

On the downside the Default settings are not able to find any feasible solution to the *dense-1* instance at all. Looking into the details, the low number of branch-and-bound nodes reported in comparison to the elapsed time indicates that the subproblems are di cult to solve for the simplex algorithm. The Emphasis setting exhibits a similar slow-down. While superior for *dense-1*, the Barrier setting has problems dealing with *terminalintens-1*. Even though an optimal solution was found, the instance could not be finished in time.

In general it can be said that the solution time is heavily dependent on the time it takes to find the optimal primal solution.

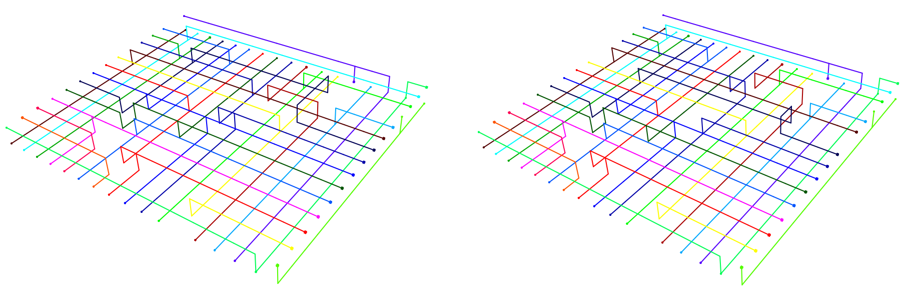
10 Martin ( ) reports that in their computations the optimal solution of a linear program has never been a feasible solution of the integer program.

6.4.3 Results for the node disjoint multi-aligned-layer model

Table . shows results for the node disjoint multi-aligned-layer model. Since this is a multi-layer model we have to assign costs to the vias. These are given in the column labeled *Via-cost*. The next three columns list the numbers of vias, "regular" arcs, and vias+arcs in the optimal solution.

In case of unit via costs, the objective value of the relaxation is equal to the objective value of the optimal integer solution for all instances except for *moredifficult-2*. The value of the relaxation for *moredifficult-2* is . . This is weaker than the value reported in Grötschel et al. ( )[11], while for *pedabox-2* the relaxation is stronger than reported. Note that in five out of seven instances with unit via costs the Barrier setting gives the best performance.

To our knowledge this is the first time that Manhattan solutions are computed for the *dense* (Luk,) and *modifieddense* (Coohoon and Heck,) problems. As reported in Grötschel et al. ( ), both problems are not solvable with the Manhattan one-layer model and have therefore no Manhattan solution in two layers. As can be seen in Figures . a and . b both problems have a three-layer solution, with only one net (dark blue at three o'clock) using the third layer at a single point.



(a) dense-

(b) modifieddense-

Figure . : Node disjoint three-aligned-layer solutions

Via minimization

Traditionally via minimization is viewed as a separate problem after the routing has taken place (Grötschel et al.,). Since we work with multi-layer models via minimization is part of the routing. As can be seen in Table . we tried the "classical" instances with three di erent cost settings for the vias. First unit costs were used to minimize the total number of arcs, including vias. Next, the number of vias was minimized by setting the cost to , , which is above the total cost of all "regular" arcs,

11 This indicates that some of the strengthening cuts used by Grötschel et al. ( ) to tighten the undirected partitioning formulation can also be used to tighten the directed flow formulation.

Name	CS	B&B Nodes	Time [s]	Via-cost	Vias	Arcs	Vias +Arcs
augmenteddense-2	B	1	60	1	35	469	504
augmenteddense-2	D	1	120	1	35	469	504
augmenteddense-2	E	1	117	1	35	469	504
augmenteddense-2	B	1	261	1000	35	469	504
augmenteddense-2	B	1	372	0.001	35	469	504
difficult-2	B	1	43	1	56	470	526
difficult-2	D	1	276	1	56	470	526
difficult-2	E	1	274	1	56	470	526
difficult-2	B	9	817	1000	51	484	535
difficult-2	B	11	1,083	0.001	63	469	532
moredifficult-2	B	25	863	1	61	461	522
moredifficult-2	D	525	22,712	1	60	462	522
moredifficult-2	E	14	1071	1	61	461	522
moredifficult-2	B	74	4,502	1000	53	481	534
moredifficult-2	B	3	395	0.001	61	461	522
pedabox-2	B	1	14	1	47	343	390
pedabox-2	D	1	52	1	47	343	390
pedabox-2	E	1	52	1	47	343	390
pedabox-2	B	17	486	1000	47	343	390
pedabox-2	B	14	391	0.001	47	343	390
terminalintens-2	B	1	139	1	59	537	596
terminalintens-2	D	1	58	1	59	537	596
terminalintens-2	E	1	54	1	59	537	596
terminalintens-2	B	1	62	1000	55	562	617
terminalintens-2	B	1	478	0.001	59	537	596
dense-3	B	1	161	1	35	436	471
dense-3	D	1	127	1	35	436	471
dense-3	E	1	125	1	35	436	471
dense-3	B	1	204	1000	35	436	471
dense-3	B	1	610	0.001	35	436	471
modifieddense-3	B	1	184	1	35	450	485
modifieddense-3	D	1	308	1	35	450	485
modifieddense-3	E	1	311	1	35	450	485
modifieddense-3	B	1	448	1000	35	450	485
modifieddense-3	B	1	579	0.001	35	450	485

Table . : Results for the node disjoint multi-aligned-layer model (part) ensuring that a global minimum is reached. Finally, the cost of each via was set to . which is equal to minimizing the number of "regular" arcs. This results in solutions that have the same number of arcs as reported by Grötschel et al. ( ) for the Manhattan one-layer model.

Interestingly, the number of vias is constant for *augmenteddense-2, pedabox-2, modifieddense-3,* and *dense-3*. For the other instances, minimization of the number of vias always results in detours, i. e., higher total number of arcs used.

Performance

Grötschel et al. ( ) report solution times for the Manhattan one-layer model on a / with megahertz. Of course, any comparison of times between different processors is highly inaccurate and debatable. Nevertheless, we will make some educated guesses. The results for the node disjoint two-aligned-layer model in Table . were computed on a , megahertz computer. This gives us a factor of . If we compare our best solution times with the ones reported, the geometric mean of the speed-up for all five solvable instances is , . This is nearly twenty times faster than what we would have expected from the megahertz figure. Furthermore, this is the comparison between a special purpose code with preprocessing, separation routines and problem specific primal heuristics with a *generate the whole model and feed it into a- tandard solver* approach without any problem specific routines. We can conclude from the value of the root relaxation that the partitioning formulation with additional strengthening cuts and the directed multicommodity flow formulation are about equally strong in practice. It should be noted, though, that for *moredifficult-2*, the only instance where the flow formulation is weaker, we also have the least improvement by only a factor of , while for *pedabox-2*, the only instance where the flow formulation is stronger, we have the highest improvement by a factor of , . The rest of the speed-up seems to come from .[12] The numbers are compatible with those given in Bixby et al. ( ) and Bixby ( ), keeping in mind that the improvement in hardware speed of times is only a gross approximation.

New instances

All the instances presented so far are quite old and can be solved in less than one hour. To get an outlook on how far our approach will take us, we tried a few new instances. The results can be found in Table . .

sb40-56, sb3-30-26d, and *sb11-20-7* are all random generated switchbox instances. *sb40-56* is about four times the size of the "classical" instances and the resulting has more than three million non-zero entries in the constraint matrix. Regarding memory consumption, this is on the limit what can be solved in two gigabytes of with . *sb11-20-7* is noteworthy because all nets have eleven terminals. This value is substantially higher compared to the "classical" instances where the nets have at most six terminals.

12 Since we use a di erent model, part of the speed-up might possibly be due to the model being more amenable for the solver.

Name	CS	B&B Nodes	Time [s]	Via-cost	Vias	Arcs	Vias +Arcs
sb11-20-7	B	1	16,437	1	107	486	593
	E	1	65,393	1	107	486	593
sb3-30-26d	B	1	1,455	1	130	1286	1416
	E	1	47,335	1	130	1286	1416
sb40-56	B	1	3,846	1	166	2286	2452
	D	1	518	1	166	2286	2452
	E	3	776	1	166	2286	2452
taq-3	B	71	931	1	66	371	437
	D	4	385	1	66	371	437
	E	19	346	1	66	371	437
alue-4	B	124	18,355	1	117	668	785
	D	1	3,900	1	117	668	785
	E	4	1,825	1	117	668	785

Table . : Results for the node disjoint multi-aligned-layer model (part)

For all three instances the value of the relaxation is equal to the value of the integer optimal solution. Pictures of the solutions can be found in Figures . , . and . .

Solving the root relaxation for *sb11-20-7* with the dual simplex algorithm took more than hours. Interestingly, the solution was immediately integer feasible. For *sb3-30-26d* it took more than hours to solve the root relaxation, but again the result was immediately integer feasible.

When comparing the timings it should be kept in mind that the number of branch-and-bound nodes is not equal to the number of linear programs solved.[13] When using the Barrier setting, even though no branching was performed, eleven linear programs had to be solved to compute the solution to *sb11-20-7*, none were needed for *sb3-30-26d* and s had to be solved for *sb40-56*.

taq-3 (Figure .) and *alue-4* (Figure .) are general routing problems based on circuits described in Jünger et al. ( ). *alue-4* is the only instance so far that requires four aligned layers. In both instances the relaxation does not reach the value of the optimum integer solution. For *taq-3* the relaxation objective value is , and for *alue-4* it is . .

We also generated one instance of a general routing problem in the node disjoint two-crossed-layer model, namely *gr2-8-32*. The forbidden area of the graph is located only in one layer, as can be seen in the left half of Figure . . The optimal solution is shown in the right half of Figure . . The objective value is , including vias of unit cost. The objective value of the root relaxation is . Using the Emphasis setting the solution time was , seconds. , branch-and-bound nodes were generated.

13 Rounding and diving heuristics, for example, also use linear programs.

6.5 Outlook

From the results shown it became clear that the approach presented in this chapter has not reached its limits yet. Several complementary improvements are possible:

- Use of problem specific preprocessing, especially the computation of node disjoint critical cuts to reduce the number of variables.
- Use of parallel computers with much memory. The barrier algorithm can utilize several processors and also branching can be done in parallel. More memory is evidently needed for larger instances.
- Further strengthening of the formulation. Very often the relaxation already yields the value of the optimal integer solution but is not feasible. Can this be improved by either problem specific valid inequalities or Gomory cuts? If there is a gap, can classes of violated inequalities be found to improve this?
- If the relaxation is non-integral, but has an equal objective value as the optimal integer solution, using a pivot and complement type heuristics like those described in Balas and Martin ( ), Balas et al. ( ) seem promising.

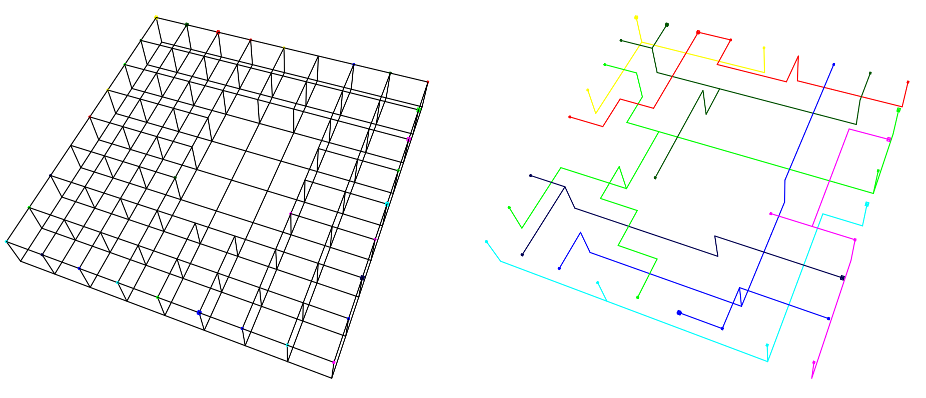


Figure . : gr - -

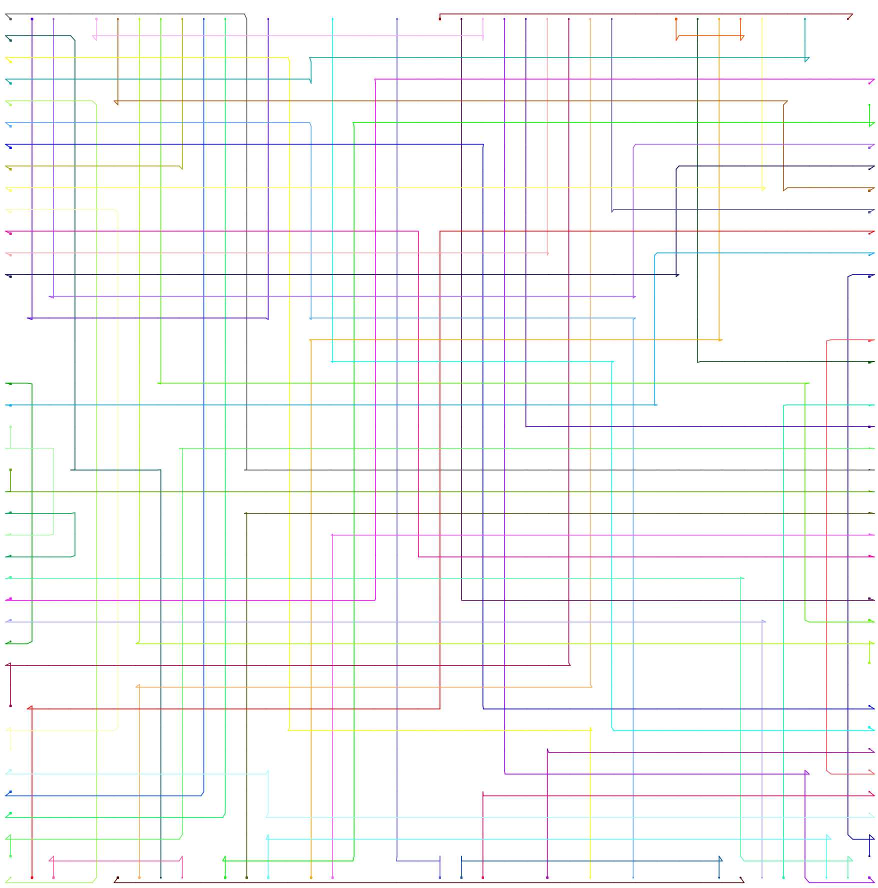


Figure . : sb -

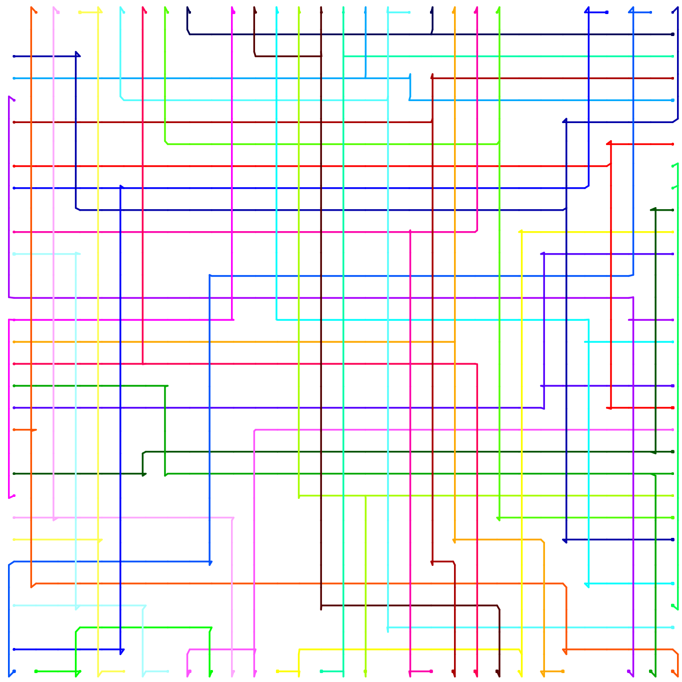


Figure . :
sb - - d

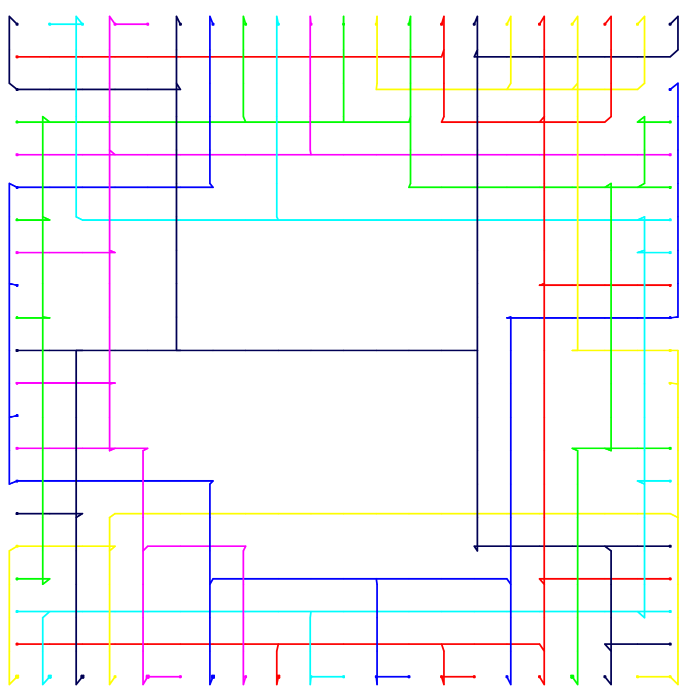


Figure . :
sb - -

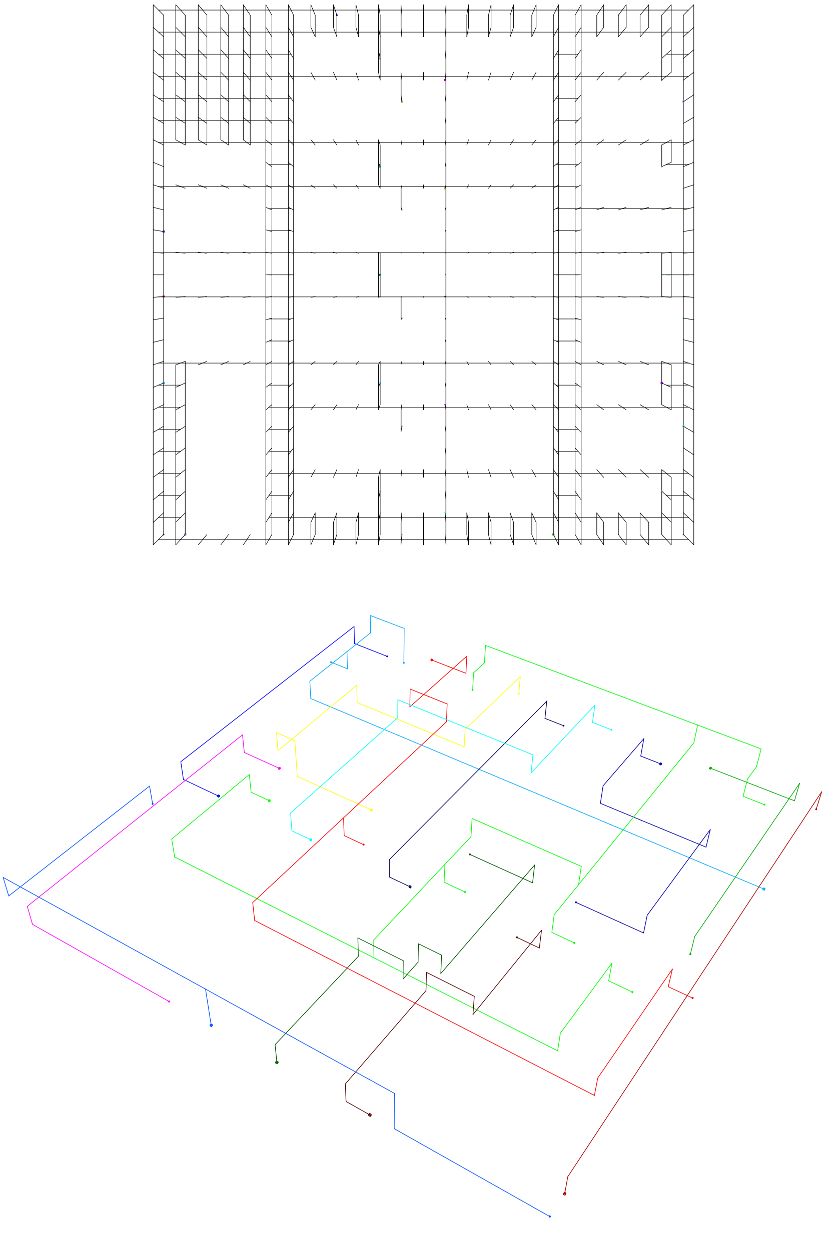


Figure . : taq-

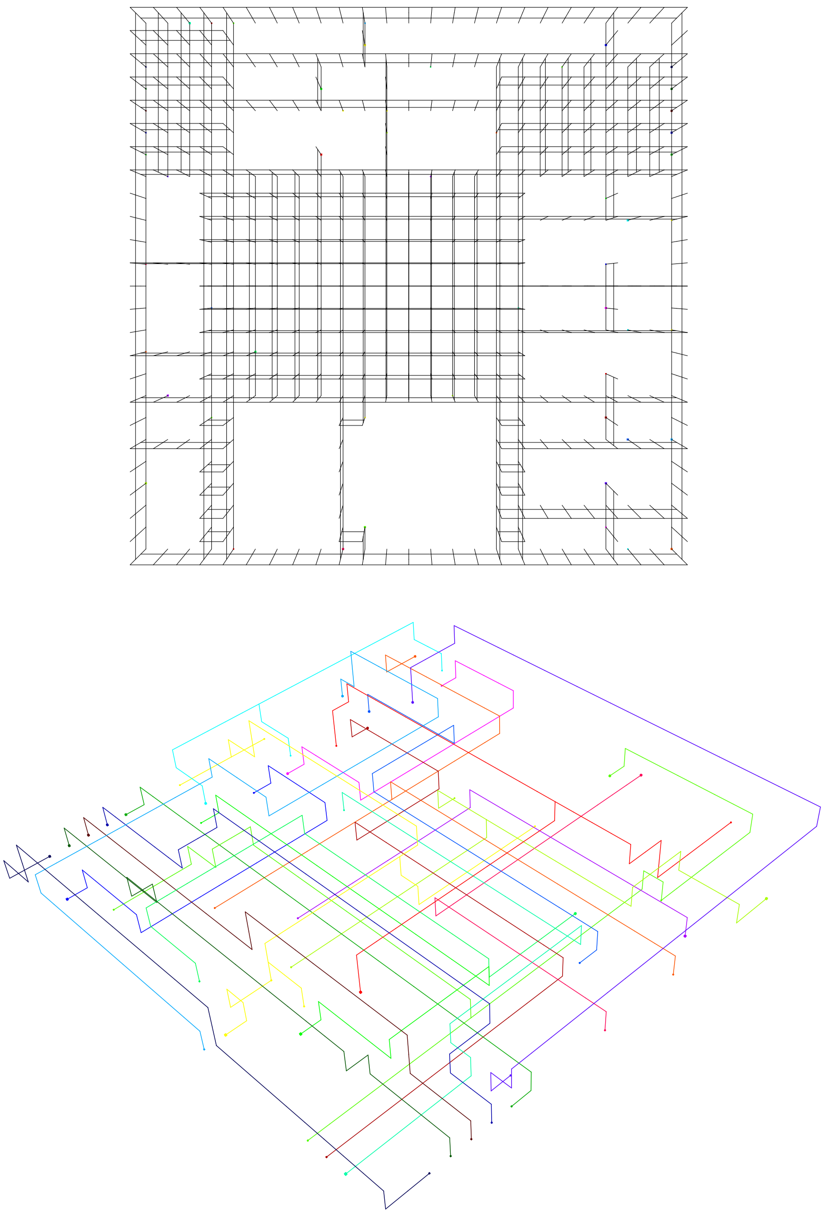


Figure . : alue-

Chapter 7

Perspectives

The problem with engineers is that they tend to cheat in order to get results.
The problem with mathematicians is that they tend to work on toy problems in order to get results.
The problem with program verifiers is that they tend to cheat at toy problems in order to get results
— fortune( )

We have seen that modeling languages together with standard solvers are a winning combination to solve many real-world (and even some mathematical) problems. Regarding the real-world it turned out that understanding the problem itself and the limitations presented by the available data are often a bigger obstacle than building and solving the mathematical model itself. Especially looking at Chapter one could ask: *What is it about? The models presented can be written down and solved in a few hours.*

But this is precisely our point. The goal was to make it easy. Problems that a few years ago (if not today) were solved by implementing special tailored branch-and-cut codes can now be tackled within days. Maybe the hand-made branch-and-cut code employing problem specific heuristics and separators would be a bit faster, but how fast has it to be in order to make up for the development time. And maybe it is not faster at all, because the latest stand-alone state-of-the art solver may, for example, use more sophisticated branching- and variable-selection rules.

Doing mathematical computations on a high level of abstraction makes it also easier to reproduce the results. One might ask what another person would have to do, to reproduce the work done in Chapter ? He or she would have to

i) take the descriptions of the problems from Appendix D,

ii) write down the model given in Section . . in any suitable modeling language,

iii) write a little program to prepare the data from the descriptions, and

iv) use a standard out-of-the-box solver to solve the instances.

This could be done in a few days. No special knowledge of combinatorics, polyhedra, or how to implement branch-and-cut algorithms is needed.

The use of extended functions in modeling languages makes it even easier to turn complex problems into models. It seems likely that future solvers will "understand" these extended functions directly and convert them themselves to whatever suits them best.

We hope that the current trend to produce open-source software persists and gets stronger in the mathematical community. In a recent interview for Business Week Linus Torvalds said

> *I compare it to science vs. witchcraft.*
> *In science, the whole system builds on people looking at other people's results and building on top of them. In witchcraft, somebody had a small secret and guarded it—but never allowed others to really understand it and build on it. Traditional software is like witchcraft. In history, witchcraft just died out. The same will happen in software. When problems get serious enough, you can't have one person or one company guarding their secrets. You have to have everybody share in knowledge.*

While some skepticism seems advisable about this forecast, science certainly has a lot to loose if the software it depends on more and more is not publicly available. This also extends to data. Many papers are published claiming in their introduction practical applications. Interestingly most of them do not deal with real-world data. And those which do, usually do not publish it.

There is another reason why sharing knowledge is so important. The software we build is getting more complex all of the time. Right now it gets increasingly visible that the industry with their "witchcraft" approach is having more and more problems to control this complexity. It is getting a commonplace experience that devices malfunction due to software problems. Z makes things simpler.

Z is not a toy. It can generate very complex models, like for example the snapshot model shown in Appendix C. . It needs less than seconds time to generate the *sb40-56* instance with more than one million variables and three million non-zero entries and it has been used successfully in several classes and projects.

We hope to have made our tiny but useful contribution to the store of publicly available software and have at least set an example to make mathematical experiments easier and more reproducible.

Appendix A

Notation

A (simple undirected) *graph* $G = (V, E)$ consists of a finite nonempty set V of *nodes* (or *vertices*) and a finite set $E \subseteq V \times V$ of *edges*. With every edge, an unordered pair of nodes, called its endnodes, is associated and we say that an edge is *incident* to its endnodes. We denote an edge e with endnodes i and j by (i, j). We assume that the two endnodes of an edge are distinct, i. e., we do not allow loops, unless specified otherwise. Two edges are called parallel if they have the same endnodes. A graph without parallel edges is called *simple*. If not otherwise noted, we always assume simple graphs.

We call a graph G a *complete rectangular* $h \times w$ *grid graph*, if it can be embedded in the plane by h horizontal lines and w vertical lines such that the nodes V are represented by the intersections of the lines and the edges are represented by the connections of the intersections. A *grid graph* is a graph that is obtained from a complete rectangular grid graph by deleting some edges and removing isolated nodes, i. e., nodes that are not incident to any edge.

A (simple) *directed graph* (or *digraph*) $D = (V, A)$ consists of a finite nonempty set V of *nodes* (or *vertices*) and a set A of *arcs*. With every arc a, an ordered pair (u, v) of nodes, called its *endnodes*, is associated; u is the *initial endnode* (or *tail*) and v the *terminal endnode* (or *head*) of a. As in the undirected case, loops (u, u) will only be allowed if explicitly stated. We denote an arc a with tail u and head v by (u, v); we also say that a *goes from* u *to* v, that a is *incident from* u and *incident to* v, and that a *leaves* u and *enters* v.

If A is a real $m \times n$ matrix and $b \in \mathbb{R}^m$, then $Ax \leqslant b$ is called a *system of (linear) inequalities*, and $Ax = b$ a *system of (linear) equations*. The solution set $\{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ of a system of inequalities is called a *polyhedron*. A polyhedron P that is *bounded* is called a *polytope*.

We call finding a vector $x^* \in P = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ maximizing the linear function $c^T x$ over P for a given $m \times n$ matrix A, a vector $b \in \mathbb{R}^m$, and a vector $c \in \mathbb{R}^n$, a linear programming problem or short *linear program* or . We usually just write $\max c^T x$ subject to $Ax \leqslant b$.

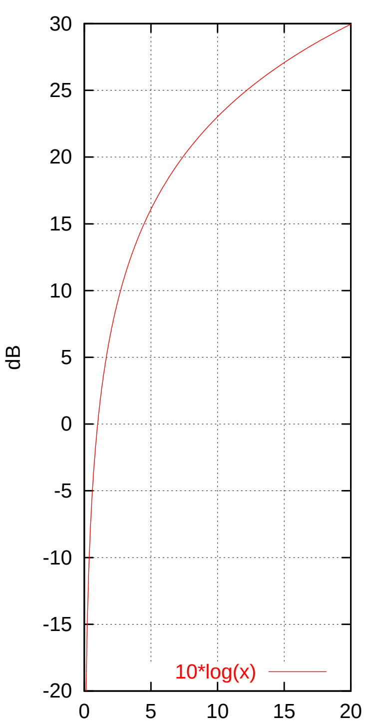
We call finding an integral vector $x^* \in P = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ maximizing the linear function $c^T x$ over the integral vectors in P for a given an $m \times n$ matrix A, a vector $b \in \mathbb{R}^m$ and a vector $c \in \mathbb{R}^n$, an integer linear programming problem, or short *integer program* or . Given an integer linear program, the linear program which arises by dropping the integrality constraints is called its *relaxation*.

A.1 Decibel explained

A *Bel* (symbol B) is a dimensionless unit of measure of ratios, named in honor of Alexander Graham Bell. The decibel (dB), or one-tenth of a bel is defined as

$$\text{decibels} = 10 \log_{10}(\text{ratio}) \quad .$$

Zero decibel mean the strength of the signal is unchanged, negative values indicate losses and positive values indicate gains. A doubling in signal power is equal to about three decibel. Since decibel describe ratios it is well suited to express losses and gains regarding signal strength. In this case the ratio is between the strength of the modified signal and the strength of the original signal. For example, if the pathloss between two points weakens a signal by a factor of $1/10^8$ this can be expressed as -80 dB.



Decibels can also be used to measure absolute power values by expressing the ratio to a known predefined power level. If the reference power level is one milliwatt, the unit is called decibel-milliwatt (dBm). For example, a signal power of 20 W equals about 43 dBm, since

$$10 \log_{10}\left(\frac{20}{0.001}\right) \approx 43 \quad .$$

Now using decibels, it is easy to add-up gains and losses. For example, a signal with a strength of 43 dBm is emitted from a base station. On its way to the mobile it is amplified by a 12 dB antenna-gain and then weakened by a pathloss of -80 dB. As a result, the mobile receives a signal with a strength of $43 + 12 - 80 = -25$ dBm. This equals about . milliwatts, as

$$20000 \cdot 15.8 \cdot 10^{-8} \approx 10^{\frac{-25}{10}} \approx 0.00316 \quad .$$

Appendix B

Zimpl Internals

B.1 The grammar of the Zimpl parser

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                    */
/*   File....: mmlparse.y                                             */
/*   Name....: MML Parser                                             */
/*   Author..: Thorsten Koch                                          */
/*   Copyright by Author, All rights reserved                         */
/*                                                                    */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
%union
{
   unsigned int bits;
   Numb*        numb;
   const char*  strg;
   const char*  name;
   Symbol*      sym;
   Define*      def;
   CodeNode*    code;
};
%token DECLSET DECLPAR DECLVAR DECLMIN DECLMAX DECLSUB
%token DEFNUMB DEFSTRG DEFSET PRINT CHECK BINARY INTEGER REAL
%token ASGN DO WITH IN TO BY FORALL EMPTY_TUPLE EMPTY_SET EXISTS
%token PRIORITY STARTVAL DEFAULT AND OR XOR NOT SUM MIN MAX
%token CMP_LE CMP_GE CMP_EQ CMP_LT CMP_GT CMP_NE INFTY
%token IF THEN ELSE END INTER UNION CROSS SYMDIFF WITHOUT PROJ
%token MOD DIV POW FAC READ AS SKIP USE COMMENT VIF VABS
%token CARD ABS SGN FLOOR CEIL LOG LN EXP SQRT RANDOM ORD
%token SUBSETS INDEXSET POWERSET
%token <sym> NUMBSYM STRGSYM VARSYM SETSYM
%token <def> NUMBDEF STRGDEF SETDEF DEFNAME
%token <name> NAME
%token <strg> STRG
%token <numb> NUMB
%token <bits> SCALE
%token <bits> SEPARATE
```

```
%type <code> stmt decl_set decl_par decl_var decl_obj decl_sub command
%type <code> def_numb def_strg def_set exec_do constraint vbool
%type <code> cexpr cexpr_list cfactor cproduct symidx tuple tuple_list
%type <code> sexpr lexpr read read_par cexpr_entry cexpr_entry_list
%type <code> set_entry idxset vproduct vfactor vexpr name_list
%type <code> set_entry_list par_default var_type con_type lower
%type <code> upper priority startval condition matrix_body matrix_head
%type <bits> con_attr con_attr_list

%right ASGN
%left  ','
%right '('
%left  ')'
%left  OR XOR
%left  EXISTS
%left  AND
%left  CMP_EQ CMP_NE CMP_LE CMP_LT CMP_GE CMP_GT
%left  IN
%left  NOT
%left  UNION WITHOUT SYMDIFF
%left  INTER
%left  CROSS
%left  '+' '-'
%left  SUM MIN MAX
%left  '*' '/' MOD DIV
%left  POW
%left  FAC
%%
stmt
   : decl_set
   | decl_par
   | decl_var
   | decl_obj
   | decl_sub
   | def_numb
   | def_strg
   | def_set
   | exec_do
   ;
decl_set
   : DECLSET NAME ASGN sexpr ';'
   | DECLSET NAME '[' idxset ']' ASGN sexpr ';'
   | DECLSET NAME '[' idxset ']' ASGN set_entry_list ';'
   | DECLSET NAME '[' ']' ASGN set_entry_list ';'
   ;
set_entry_list
   : set_entry
   | set_entry_list ',' set_entry
   | SUBSETS '(' sexpr ',' cexpr ')'
   | POWERSET '(' sexpr ')'
   ;
set_entry
   : tuple sexpr
```

```
    ;
def_numb
    : DEFNUMB DEFNAME '(' name_list ')' ASGN cexpr ';'
    ;
def_strg
    : DEFSTRG DEFNAME '(' name_list ')' ASGN cexpr ';'
    ;
def_set
    : DEFSET DEFNAME '(' name_list ')' ASGN sexpr ';'
    ;
name_list
    : NAME
    | name_list ',' NAME
    ;
decl_par
    : DECLPAR NAME '[' idxset ']'
              ASGN cexpr_entry_list par_default ';'
    | DECLPAR NAME '[' idxset ']' ASGN cexpr par_default ';'
    | DECLPAR NAME ASGN cexpr ';'
    ;
par_default
    : /* empty */
    | DEFAULT cexpr
    ;
decl_var
    : DECLVAR NAME '[' idxset ']'
                   var_type lower upper priority startval ';'
    | DECLVAR NAME '[' idxset ']' BINARY priority startval ';'
    | DECLVAR NAME var_type lower upper priority startval ';'
    | DECLVAR NAME BINARY priority startval ';'
    ;
var_type
    : /* empty */
    | REAL
    | INTEGER
    ;
lower
    : /* empty */
    | CMP_GE cexpr
    | CMP_GE '-' INFTY
    ;
upper
    : /* empty */
    | CMP_LE cexpr
    | CMP_LE INFTY
    ;
priority
    : /* empty */
    | PRIORITY cexpr
    ;
startval
    : /* empty */
    | STARTVAL cexpr
    ;
```

```
cexpr_entry_list
    : cexpr_entry
    | cexpr_entry_list ',' cexpr_entry
    | read
    | matrix_head matrix_body
    ;
cexpr_entry
    : tuple cexpr
    ;
matrix_head
    : WITH cexpr_list WITH
    ;
matrix_body
    : matrix_head cexpr_list WITH
    | matrix_body matrix_head cexpr_list WITH
    ;
decl_obj
    : DECLMIN NAME DO vexpr ';'
    | DECLMAX NAME DO vexpr ';'
    ;
decl_sub
    : DECLSUB NAME DO constraint ';'
    ;
constraint
    : vexpr con_type vexpr con_attr_list
    | vexpr con_type cexpr con_attr_list
    | cexpr con_type vexpr con_attr_list
    | cexpr con_type cexpr con_attr_list
    | cexpr con_type vexpr CMP_LE cexpr con_attr_list
    | cexpr con_type cexpr CMP_LE cexpr con_attr_list
    | cexpr con_type vexpr CMP_GE cexpr con_attr_list
    | cexpr con_type cexpr CMP_GE cexpr con_attr_list
    | FORALL idxset DO constraint
    | IF lexpr THEN constraint ELSE constraint END
    | VIF vbool THEN vexpr con_type vexpr
                ELSE vexpr con_type vexpr END
    | VIF vbool THEN cexpr con_type vexpr
                ELSE vexpr con_type vexpr END
    | VIF vbool THEN vexpr con_type cexpr
                ELSE vexpr con_type vexpr END
    | VIF vbool THEN vexpr con_type vexpr
                ELSE cexpr con_type vexpr END
    | VIF vbool THEN vexpr con_type vexpr
                ELSE vexpr con_type cexpr END
    | VIF vbool THEN cexpr con_type cexpr
                ELSE vexpr con_type vexpr END
    | VIF vbool THEN cexpr con_type vexpr
                ELSE cexpr con_type vexpr END
    | VIF vbool THEN cexpr con_type vexpr
                ELSE vexpr con_type cexpr END
    | VIF vbool THEN vexpr con_type cexpr
                ELSE cexpr con_type vexpr END
    | VIF vbool THEN vexpr con_type cexpr
                ELSE vexpr con_type cexpr END
```

```
    | VIF vbool THEN vexpr con_type vexpr
                ELSE cexpr con_type cexpr END
    | VIF vbool THEN cexpr con_type cexpr
                ELSE cexpr con_type vexpr END
    | VIF vbool THEN cexpr con_type cexpr
                ELSE vexpr con_type cexpr END
    | VIF vbool THEN cexpr con_type vexpr
                ELSE cexpr con_type cexpr END
    | VIF vbool THEN vexpr con_type cexpr
                ELSE cexpr con_type cexpr END
    | VIF vbool THEN cexpr con_type cexpr
                ELSE cexpr con_type cexpr END
    | VIF vbool THEN vexpr con_type vexpr END
    | VIF vbool THEN cexpr con_type vexpr END
    | VIF vbool THEN vexpr con_type cexpr END
    | VIF vbool THEN cexpr con_type cexpr END
    ;
vbool
    : vexpr CMP_NE vexpr
    | cexpr CMP_NE vexpr
    | vexpr CMP_NE cexpr
    | vexpr CMP_EQ vexpr
    | cexpr CMP_EQ vexpr
    | vexpr CMP_EQ cexpr
    | vexpr CMP_LE vexpr
    | cexpr CMP_LE vexpr
    | vexpr CMP_LE cexpr
    | vexpr CMP_GE vexpr
    | cexpr CMP_GE vexpr
    | vexpr CMP_GE cexpr
    | vexpr CMP_LT vexpr
    | cexpr CMP_LT vexpr
    | vexpr CMP_LT cexpr
    | vexpr CMP_GT vexpr
    | cexpr CMP_GT vexpr
    | vexpr CMP_GT cexpr
    | vbool AND vbool
    | vbool OR  vbool
    | vbool XOR vbool
    | NOT vbool
    | '(' vbool ')'
    ;
con_attr_list
    : /* empty */
    | con_attr_list ',' con_attr
    ;
con_attr
    : SCALE
    | SEPARATE
    ;
con_type
    : CMP_LE
    | CMP_GE
    | CMP_EQ
```

```
    ;
vexpr
    : vproduct
    | vexpr '+' vproduct
    | vexpr '-' vproduct
    | vexpr '+' cproduct %prec SUM
    | vexpr '-' cproduct %prec SUM
    | cexpr '+' vproduct
    | cexpr '-' vproduct
    ;
vproduct
    : vfactor
    | vproduct '*' cfactor
    | vproduct '/' cfactor
    | cproduct '*' vfactor
    ;
vfactor
    : VARSYM symidx
    | '+' vfactor
    | '-' vfactor
    | VABS '(' vexpr ')'
    | SUM idxset DO vproduct %prec '+'
    | IF lexpr THEN vexpr ELSE vexpr END
    | '(' vexpr ')'
    ;
exec_do
    : DO command ';'
    ;
command
    : PRINT cexpr
    | PRINT tuple
    | PRINT sexpr
    | CHECK lexpr
    | FORALL idxset DO command
    ;
idxset
    : tuple IN sexpr condition
    | sexpr condition
    ;
condition
    : /* empty */
    | WITH lexpr
    ;
sexpr
    : SETSYM symidx
    | SETDEF '(' cexpr_list ')'
    | EMPTY_SET
    | '{' cexpr TO cexpr BY cexpr '}'
    | '{' cexpr TO cexpr '}'
    | sexpr UNION sexpr
    | sexpr '+' sexpr %prec UNION
    | sexpr SYMDIFF sexpr
    | sexpr WITHOUT sexpr
    | sexpr '-' sexpr %prec WITHOUT
```

```
    | sexpr CROSS   sexpr
    | sexpr '*' sexpr
    | sexpr INTER   sexpr
    | '(' sexpr ')'
    | '{' tuple_list '}'
    | '{' cexpr_list '}'
    | '{' idxset '}'
    | PROJ '(' sexpr ',' tuple ')'
    | INDEXSET '(' SETSYM ')'
    | IF lexpr THEN sexpr ELSE sexpr END
    ;
read
    : READ cexpr AS cexpr
    | read read_par
    ;
read_par
    : SKIP cexpr
    | USE cexpr
    | COMMENT cexpr
    ;
tuple_list
    : tuple
    | tuple_list ',' tuple
    | read
    ;
lexpr
    : cexpr CMP_EQ cexpr
    | cexpr CMP_NE cexpr
    | cexpr CMP_GT cexpr
    | cexpr CMP_GE cexpr
    | cexpr CMP_LT cexpr
    | cexpr CMP_LE cexpr
    | sexpr CMP_EQ sexpr
    | sexpr CMP_NE sexpr
    | sexpr CMP_GT sexpr
    | sexpr CMP_GE sexpr
    | sexpr CMP_LT sexpr
    | sexpr CMP_LE sexpr
    | lexpr AND lexpr
    | lexpr OR lexpr
    | lexpr XOR lexpr
    | NOT lexpr
    | '(' lexpr ')'
    | tuple IN sexpr
    | EXISTS '(' idxset ')' %prec EXISTS
    ;
tuple
    : EMPTY_TUPLE
    | CMP_LT cexpr_list CMP_GT
    ;
symidx
    : /* empty */
    | '[' cexpr_list ']'
    ;
```

```
cexpr_list
    : cexpr
    | cexpr_list ',' cexpr
    ;
cexpr
    : cproduct
    | cexpr '+' cproduct
    | cexpr '-' cproduct
    ;
cproduct
    : cfactor
    | cproduct '*' cfactor
    | cproduct '/' cfactor
    | cproduct MOD cfactor
    | cproduct DIV cfactor
    | cproduct POW cfactor
    ;
cfactor
    : NUMB
    | STRG
    | NAME
    | NUMBSYM symidx
    | STRGSYM symidx
    | NUMBDEF '(' cexpr_list ')'
    | STRGDEF '(' cexpr_list ')'
    | cfactor FAC
    | CARD '(' sexpr ')'
    | ABS '(' cexpr ')'
    | SGN '(' cexpr ')'
    | FLOOR '(' cexpr ')'
    | CEIL '(' cexpr ')'
    | LOG '(' cexpr ')'
    | LN '(' cexpr ')'
    | EXP '(' cexpr ')'
    | SQRT '(' cexpr ')'
    | '+' cfactor
    | '-' cfactor
    | '(' cexpr ')'
    | RANDOM '(' cexpr ',' cexpr ')'
    | IF lexpr THEN cexpr ELSE cexpr END
    | MIN idxset DO cproduct %prec '+'
    | MAX idxset DO cproduct %prec '+'
    | SUM idxset DO cproduct %prec '+'
    | MIN '(' cexpr_list ')'
    | MAX '(' cexpr_list ')'
    | ORD '(' sexpr ',' cexpr ',' cexpr ')'
    ;
```

B.2 Detailed function statistics

Here are the code statistics for each function within Z . The name in bold writing is the name of the module. Aggregated statistics per module can be found in Section . . on page . The first column of the tables list the name of the functions. *Lines* denotes the number of code lines, i. e., without empty and comment lines. *Stmt.* is the number of statements in the function. *Calls* is the number of calls to other functions. *CC* is the cyclomatic complexity number. *Dp.* is the maximal nesting depth of *if*, *for*, *while*, and *do* constructs. *Ex.* is the number of *return* statements and *As.* the number of asserts. *Cover* denotes the percentage of statements that are executed by the regression tests. √ indicates %, while ⊗ means not called at all.

bound.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
bound_new	14	9	4	2	1		3	√
bound_free	8	5	4	2			1	√
bound_is_valid	6	1	1	6				√
bound_copy	5	2	2				1	√
bound_get_type	5	2	1				1	√
bound_get_value	6	3	1				2	√
6 functions, total	44	22	13	13			8	100%
∅ per function	7.3	3.7	2.2	2.2			1.3	
∅ statements per	0.5		1.7	1.7			2.4	

code.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
code_is_valid	4	1	1	2				√
code_new_inst	27	20	9	3	1		4	√
code_new_numb	13	10	5				2	√
code_new_strg	14	11	5				3	√
code_new_name	14	11	5				3	√
code_new_size	14	11	5				3	√
code_new_varclass	13	10	5				2	√
code_new_contype	13	10	5				2	√
code_new_bits	13	10	5				2	√
code_new_symbol	13	10	5				2	√
code_new_define	13	10	5				2	√
code_new_bound	14	11	6	2			3	√
code_free_value	58	29	11	18	1		1	80%
code_free	9	7	3	3				√
code_set_child	8	5	1				4	⊗
code_get_type	5	2	1				1	√
code_get_inst	5	2	1				1	√
code_set_root	5	2	1				1	√
code_get_root	4	1	0					√
code_get_inst_count	4	1	0					⊗
code_check_type	20	9	3	2	1		3	√
code_errmsg	9	4	6	2				80%
code_eval	6	3	1				1	√

(continued on next page)

code.c *(cont.)*	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
code_get_child	8	5	1				4	√
code_get_numb	4	1	1					√
code_get_strg	4	1	1					√
code_get_name	4	1	1					√
code_get_tuple	4	1	1					√
code_get_set	4	1	1					√
code_get_idxset	4	1	1					√
code_get_entry	4	1	1					√
code_get_term	4	1	1					√
code_get_size	4	1	1					√
code_get_bool	4	1	1					√
code_get_list	4	1	1					√
code_get_varclass	4	1	1					√
code_get_contype	4	1	1					√
code_get_rdef	4	1	1					√
code_get_rpar	4	1	1					√
code_get_bits	4	1	1					√
code_get_symbol	4	1	1					√
code_get_define	4	1	1					√
code_get_bound	4	1	1					√
code_value_numb	7	4	2				1	√
code_value_strg	8	5	2				2	√
code_value_name	8	5	2				2	√
code_value_tuple	8	5	3				2	√
code_value_set	8	5	3				2	√
code_value_idxset	8	5	3				2	√
code_value_entry	8	5	3				2	√
code_value_term	8	5	3				2	√
code_value_bool	7	4	2				1	√
code_value_size	7	4	2				1	⊗
code_value_list	8	5	3				2	√
code_value_varclass	7	4	2				1	⊗
code_value_contype	7	4	2				1	⊗
code_value_rdef	8	5	3				2	√
code_value_rpar	8	5	3				2	√
code_value_bits	7	4	2				1	⊗
code_value_bound	7	4	2				1	√
code_value_void	6	3	2				1	√
code_copy_value	65	40	13	19	1		2	28%
code_eval_child	4	1	2					√
code_eval_child_numb	4	1	3					√
code_eval_child_strg	4	1	3					√
code_eval_child_name	4	1	3					√
code_eval_child_tuple	4	1	3					√
code_eval_child_set	4	1	3					√
code_eval_child_idxset	4	1	3					√
code_eval_child_entry	4	1	3					√
code_eval_child_term	4	1	3					√
code_eval_child_size	4	1	3					√
code_eval_child_bool	4	1	3					√
code_eval_child_list	4	1	3					√

(continued on next page)

code.c *(cont.)*	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
code_eval_child_varclass	4	1	3					√
code_eval_child_contype	4	1	3					√
code_eval_child_rdef	4	1	3					√
code_eval_child_rpar	4	1	3					√
code_eval_child_bits	4	1	3					√
code_eval_child_symbol	4	1	3					√
code_eval_child_define	4	1	3					√
code_eval_child_bound	4	1	3					√
82 functions, total	662	355	225	125			74	85%
∅ per function	8.1	4.3	2.7	1.5			0.9	
∅ statements per	0.5		1.6	2.8			4.7	

conname.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
conname_format	4	1	0					√
conname_free	9	6	2				2	√
conname_set	19	16	8	3		3	4	√
conname_get	33	19	10	6	2		4	√
conname_next	4	1	0					√
5 functions, total	69	43	20	12			10	100%
∅ per function	13.8	8.6	4.0	2.4			2.0	
∅ statements per	0.6		2.1	3.6			3.9	

define.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
define_new	18	15	4				5	√
define_set_param	6	3	2				2	√
define_set_code	6	3	1				2	√
define_exit	14	10	4	2	1		1	√
define_is_valid	4	1	1	2				√
define_lookup	9	7	1	3			1	√
define_get_name	5	2	1				1	√
define_get_type	5	2	1				1	√
define_get_param	5	2	1				1	√
define_get_code	5	2	1				1	√
10 functions, total	77	47	17	14			15	100%
∅ per function	7.7	4.7	1.7	1.4			1.5	
∅ statements per	0.6		2.8	3.4			2.9	

elem.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
extend_storage	27	23	6	2	1		6	√
new_elem	13	10	2	2			3	√
elem_init	3	0	0					√
elem_exit	13	9	3	3	1			88%
elem_new_numb	8	5	2				1	√
elem_new_strg	9	6	1				2	√
elem_new_name	9	6	1				2	√
elem_free	10	7	2	2			1	√
elem_is_valid	4	1	1	2				√
elem_copy	14	8	4	2	1		2	√
elem_cmp	28	20	9	6	1	4	6	90%

elem.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
elem_get_type	5	2	1				1	√
elem_get_numb	6	3	1				2	√
elem_get_strg	7	4	1				3	√
elem_get_name	7	4	1				3	√
elem_print	19	8	5	4	1		1	90%
elem_hash	20	9	3	4	1			72%
elem_tostr	24	13	6	4	1		3	80%
18 functions, total	226	138	49	39			36	93%
∅ per function	12.6	7.7	2.7	2.2			2.0	
∅ statements per	0.6		2.8	3.5			3.7	

entry.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
entry_new_numb	13	10	5				3	√
entry_new_strg	14	11	4				4	√
entry_new_set	14	11	5				4	√
entry_new_var	14	11	4				4	√
entry_free	26	13	6	6	2		1	86%
entry_is_valid	4	1	1	2				√
entry_copy	7	4	1				1	√
entry_cmp	6	3	3				2	√
entry_get_type	5	2	1				1	√
entry_get_tuple	6	3	2				2	√
entry_get_numb	6	3	1				2	√
entry_get_strg	6	3	1				2	√
entry_get_set	6	3	1				2	√
entry_get_var	6	3	1				2	√
entry_print	25	12	8	6	1		1	57%
15 functions, total	158	93	44	26			31	92%
∅ per function	10.5	6.2	2.9	1.7			2.1	
∅ statements per	0.6		2.1	3.6			2.9	

gmpmisc.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
pool_alloc	20	16	1	3	1		1	√
pool_free	6	3	0					√
pool_exit	10	6	1	2	1			√
gmp_str2mpq	50	34	7	10	2		2	85%
gmp_print_mpq	11	8	7					⊗
gmp_alloc	6	3	2	2		2		√
gmp_realloc	25	18	7	5	1	4	4	68%
gmp_free	7	3	2	2				√
gmp_init	11	8	7	2				88%
gmp_exit	7	4	4					√
10 functions, total	153	103	38	29			7	80%
∅ per function	15.3	10.3	3.8	2.9			0.7	
∅ statements per	0.7		2.7	3.6			12.9	

hash.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
hash_new	26	18	4	3	1		5	√
hash_free	20	15	6	4	2		1	92%
hash_is_valid	6	1	1	5				√
hash_add_tuple	14	11	4				4	√
hash_add_entry	16	13	5				4	√
hash_has_tuple	11	9	4	3			2	√
hash_has_entry	11	9	4	3			2	√
hash_lookup_entry	15	13	5	4		2	4	√
hash_add_elem_idx	14	11	4				3	√
hash_lookup_elem_idx	14	12	4	4		2	3	√
hash_statist	35	29	2	7	1		3	⊗
11 functions, total	182	141	43	36			31	79%
∅ per function	16.5	12.8	3.9	3.3			2.8	
∅ statements per	0.8		3.3	3.9			4.4	

idxset.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
idxset_new	15	12	7				5	√
idxset_free	8	5	5				1	√
idxset_is_valid	4	1	1	2				√
idxset_copy	5	2	2				1	⊗
idxset_get_lexpr	5	2	1				1	√
idxset_get_tuple	5	2	1				1	√
idxset_get_set	5	2	1				1	√
idxset_is_unrestricted	5	2	1				1	√
idxset_print	10	7	7				1	⊗
9 functions, total	62	35	26	10			12	75%
∅ per function	6.9	3.9	2.9	1.1			1.3	
∅ statements per	0.6		1.3	3.5			2.7	

inst.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
i_nop	8	5	4	2			1	√
i_subto	17	12	9	2	1		1	√
i_constraint	46	32	23	8	2		3	√
i_rangeconst	53	35	37	6	2		1	√
i_forall	27	22	16	3	1		1	√
i_expr_add	8	4	6				1	√
i_expr_sub	8	4	6				1	√
i_expr_mul	8	4	6				1	√
i_expr_div	16	10	10	2	1		1	√
i_expr_mod	16	10	10	2	1		1	√
i_expr_intdiv	17	10	10	2	1		1	70%
i_expr_pow	20	14	12	2	1		1	√
i_expr_neg	10	7	6				1	√
i_expr_abs	10	7	6				1	√
i_expr_sgn	10	7	6				1	√
i_expr_floor	10	7	6				1	√
i_expr_ceil	10	7	6				1	√
i_expr_log	14	9	6	2	1		1	√
i_expr_ln	14	9	6	2	1		1	√

(continued on next page)

inst.c *(cont.)*	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
i_expr_sqrt	14	9	6	2	1		1	77%
i_expr_exp	7	4	5				1	√
i_expr_fac	31	22	15	4	1		1	√
i_expr_card	9	6	6				1	√
i_expr_rand	5	1	0					⊗
i_expr_if	10	6	7	2			1	√
i_expr_min	42	31	22	6	3		1	√
i_expr_max	42	31	22	6	3		1	√
i_expr_sum	28	23	18	3	1		1	√
i_expr_min2	37	27	15	5	2		2	√
i_expr_max2	37	27	15	5	2		2	√
i_expr_ord	62	45	36	9	1		1	97%
i_bool_true	7	4	3				1	√
i_bool_false	7	4	3				1	⊗
i_bool_not	7	4	4				1	√
i_bool_and	8	4	5	2			1	√
i_bool_or	8	4	5	2			1	√
i_bool_xor	11	8	5	4			1	⊗
i_bool_eq	39	26	18	5	1		2	96%
i_bool_ne	7	4	5				1	√
i_bool_ge	39	26	18	5	1		2	88%
i_bool_gt	39	26	18	5	1		2	88%
i_bool_le	7	4	5				1	√
i_bool_lt	7	4	5				1	√
i_bool_seq	11	8	6				1	√
i_bool_sneq	11	8	6				1	√
i_bool_subs	11	8	6				1	√
i_bool_sseq	11	8	6				1	√
i_bool_is_elem	11	8	6				1	⊗
i_bool_exists	27	22	15	3	1		1	⊗
i_set_new_tuple	29	22	16	3	2		2	√
i_set_new_elem	7	4	5				1	√
i_set_pseudo	7	4	4				1	⊗
i_set_empty	9	6	5				1	√
i_set_union	17	12	10	2	1		1	√
i_set_minus	17	12	10	2	1		1	√
i_set_inter	17	12	10	2	1		1	√
i_set_sdiff	17	12	10	2	1		1	√
i_set_cross	11	8	6				1	√
i_set_range	54	43	29	6	1		1	√
i_set_proj	48	36	24	6	2		1	√
i_set_indexset	10	7	6				2	√
i_tuple_new	14	12	9	2			1	√
i_tuple_empty	7	4	4				1	√
set_from_idxset	64	38	27	7	4		7	√
i_newsym_set1	37	28	22	3	1		1	√
i_newsym_set2	67	49	35	6	2		2	√
i_newsym_para1	83	62	45	9	2		3	√
i_newsym_para2	73	53	39	9	2		1	96%
i_newsym_var	136	91	101	24	4		6	80%

(continued on next page)

inst.c *(cont.)*	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
i_symbol_deref	42	28	23	6	1		2	96%
i_newdef	11	8	9				1	√
i_define_deref	43	29	27	5	2		3	√
i_set_idxset	8	5	4					√
i_idxset_new	73	52	39	9	5		2	√
i_idxset_pseudo_new	13	10	9				1	√
i_local_deref	26	14	11	4	2		1	93%
i_term_coeff	12	9	7				1	√
i_term_const	12	9	7				1	√
i_term_add	11	8	6				1	√
i_term_sub	11	8	6				1	√
i_term_sum	29	24	18	3	1		1	√
i_term_expr	12	9	6				1	√
i_entry	31	19	15	5	1		1	95%
i_elem_list_new	24	14	13	4	1		1	93%
i_elem_list_add	28	18	15	4	1		1	94%
i_tuple_list_new	9	4	5				1	√
i_tuple_list_add	12	9	7				1	√
i_entry_list_new	9	4	5				1	√
i_entry_list_add	12	9	7				1	√
i_entry_list_subsets	40	29	20	5	1		2	√
i_entry_list_powerset	24	20	9	3	1		3	√
i_list_matrix	93	70	35	9	3		8	95%
i_matrix_list_new	10	7	7				1	√
i_matrix_list_add	11	8	9				1	√
objective	19	14	11	2	1		1	80%
i_object_min	7	4	3				1	√
i_object_max	7	4	3				1	√
i_print	36	23	20	7	1		1	66%
i_bound_new	9	4	5				1	√
i_check	14	9	7	2	1		1	√
100 functions, total	2336	1624	1282	297			135	93%
∅ per function	23.4	16.2	12.8	3.0			1.4	
∅ statements per	0.7		1.3	5.5			11.9	

iread.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
i_read_new	11	8	6				1	√
i_read_param	12	9	7				1	√
i_read_comment	9	6	5				1	√
i_read_use	25	18	13	3	1		1	88%
i_read_skip	25	18	13	3	1		1	88%
parse_template	65	45	23	15	2		6	√
split_fields	52	34	2	15	2			94%
i_read	162	111	68	23	5		3	82%
8 functions, total	361	249	137	62			14	89%
∅ per function	45.1	31.1	17.1	7.8			1.8	
∅ statements per	0.7		1.8	4.0			16.6	

list.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
list_add_data	13	10	2				3	√
list_new	15	12	4				3	√
list_new_elem	7	4	3				1	√
list_new_tuple	7	4	3				1	√
list_new_entry	7	4	3				1	√
list_new_list	7	4	3				1	√
list_free	36	21	8	8	3		1	95%
list_is_valid	4	1	1	3				√
list_is_elemlist	5	2	1				1	√
list_is_entrylist	5	2	1				1	√
list_is_tuplelist	5	2	1				1	√
list_copy	7	4	1				1	√
list_add_elem	9	6	4				3	√
list_add_tuple	9	6	4				3	√
list_add_entry	9	6	4				3	√
list_add_list	9	6	4				3	√
list_get_elems	5	2	1				1	√
list_get_data	10	7	1	3		2	1	√
list_get_elem	8	5	2	2			2	√
list_get_tuple	8	5	2	2			2	√
list_get_entry	8	5	2	2			2	√
list_get_list	8	5	2	2			2	√
list_print	25	13	5	6	2			⊗
23 functions, total	226	136	62	43			37	90%
∅ per function	9.8	5.9	2.7	1.9			1.6	
∅ statements per	0.6		2.2	3.2			3.6	

load.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
get_line	54	37	6	18	3	2	2	94%
make_pathname	19	11	5	3	1		3	⊗
add_stmt	34	28	14	11	1		3	√
3 functions, total	107	76	25	32			8	78%
∅ per function	35.7	25.3	8.3	10.7			2.7	
∅ statements per	0.7		3.0	2.4			8.4	

local.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
local_new	11	8	2	2			2	√
local_new_frame	4	1	1					√
local_drop_frame	16	11	2	4	1			√
local_lookup	9	7	1	4			1	√
local_install_tuple	21	15	12	3	2		4	√
local_print_all	15	8	4	3	2			37%
local_tostrall	40	28	8	6	2		3	89%
7 functions, total	116	78	30	23			10	90%
∅ per function	16.6	11.1	4.3	3.3			1.4	
∅ statements per	0.7		2.6	3.4			7.1	

numbgmp.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
extend_storage	25	21	6	2	1		6	√
numb_init	8	5	3					√
numb_exit	17	12	6	3	1			91%
numb_new	13	10	3	2			1	√
numb_new_ascii	7	4	2				1	√
numb_new_integer	7	4	2				1	√
numb_new_mpq	7	4	2				1	√
numb_free	9	6	3				1	√
numb_is_valid	4	1	1	2				√
numb_copy	8	5	4				2	√
numb_equal	6	3	3				2	√
numb_cmp	6	3	3				2	√
numb_set	6	3	3				2	√
numb_add	6	3	3				2	√
numb_new_add	9	6	4				3	√
numb_sub	6	3	3				2	√
numb_new_sub	9	6	4				3	√
numb_mul	6	3	3				2	√
numb_new_mul	9	6	4				3	√
numb_div	6	3	3				2	⊗
numb_new_div	9	6	4				3	√
numb_intdiv	11	8	9				2	⊗
numb_new_intdiv	14	11	10				3	√
numb_mod	18	15	16				2	⊗
numb_new_mod	21	18	17				3	√
numb_new_pow	19	15	5	4	1		2	√
numb_new_fac	12	9	5				2	√
numb_neg	5	2	2				1	√
numb_abs	5	2	2				1	√
numb_sgn	19	8	5	4	1		1	90%
numb_get_sgn	6	2	2				1	√
numb_ceil	9	6	7				1	√
numb_floor	9	6	7				1	√
numb_new_log	15	10	9	2	1	2	1	√
numb_new_sqrt	15	10	9	2	1	2	1	81%
numb_new_exp	7	4	5				1	√
numb_new_ln	15	10	9	2	1	2	1	√
numb_todbl	5	2	2				1	√
numb_get_mpq	5	2	2				1	√
numb_print	5	2	3				1	√
numb_hash	16	9	1		2			√
numb_tostr	9	6	4				2	⊗
numb_zero	4	1	0					√
numb_one	4	1	0					√
numb_minusone	4	1	0					√
numb_is_int	11	4	4	3	1	2		√
numb_toint	6	3	4				2	√
numb_is_number	22	15	3	9		7		31%
48 functions, total	474	299	211	72			70	84%
∅ per function	9.9	6.2	4.4	1.5			1.5	
∅ statements per	0.6		1.4	4.2			4.2	

prog.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
prog_new	12	9	4				2	√
prog_free	11	9	5	2			2	√
prog_is_valid	4	1	1	2				√
prog_is_empty	4	1	0					√
prog_add_stmt	17	11	3	2	1		5	66%
prog_print	8	6	3	2			1	⊗
prog_execute	17	12	11	3	1		1	91%
7 functions, total	73	49	27	13			11	79%
∅ per function	10.4	7.0	3.9	1.9			1.6	
∅ statements per	0.7		1.8	3.8			4.1	

rathumwrite.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
write_name	43	25	8	9	4		3	69%
write_lhs	18	7	2	3	1		2	88%
write_rhs	24	13	7	4	1		2	83%
write_row	25	17	10	6	2		2	√
hum_write	110	81	43	32	2		2	81%
5 functions, total	220	143	70	54			11	79%
∅ per function	44.0	28.6	14.0	10.8			2.2	
∅ statements per	0.7		2.0	2.6			11.9	

ratlpfwrite.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
write_rhs	21	10	6	4	1		2	83%
write_row	20	15	8	5	1		3	√
lpf_write	132	97	49	38	3		3	89%
3 functions, total	173	122	63	47			8	89%
∅ per function	57.7	40.7	21.0	15.7			2.7	
∅ statements per	0.7		1.9	2.6			13.6	

ratlpstore.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
hash_valid	4	1	0	3				√
hashit	9	6	1	2			1	√
lps_hash_new	12	9	3				3	√
lps_hash_free	18	12	4	3	2		1	√
hash_lookup_var	13	11	3	4			3	√
hash_lookup_con	13	11	3	4			3	√
hash_add_var	15	12	4				5	√
hash_del_var	21	18	4	4			5	⊗
hash_add_con	15	12	4				5	√
hash_del_con	21	18	4	4			5	94%
hash_statist	35	29	2	7	1		3	⊗
lps_storage	26	20	5	2	1		4	√
lps_alloc	28	25	5				3	√
lps_free	55	50	25	10	1		1	91%
lps_number	21	16	1	3	1		7	√
lps_getvar	11	7	3	3			4	92%
lps_getcon	11	7	3	3			4	√

(continued on next page)

ratlpstore.c *(cont.)*	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
lps_getnzo	24	17	1	8	1		6	√
lps_addvar	37	31	11	2	1		6	√
lps_delvar	43	27	11	6	1		9	⊗
lps_addcon	34	28	9	2	1		6	√
lps_delcon	41	25	9	6	1		9	88%
lps_addnzo	40	30	4	4	1		9	√
lps_delnzo	26	20	2	7			3	95%
lps_setval	5	2	1				1	√
lps_getval	5	2	1				1	√
lps_setdir	5	2	1				1	√
lps_setprobname	8	5	2	2			2	⊗
lps_setobjname	8	5	2	2			2	83%
lps_setrhsname	8	5	2	2			2	⊗
lps_setbndname	8	5	2	2			2	⊗
lps_setrngname	8	5	2	2			2	⊗
lps_getcost	6	3	1				2	√
lps_haslower	6	3	1				2	√
lps_setcost	6	3	1				2	√
lps_getlower	6	3	1				2	√
lps_setlower	14	8	2	6	1		3	66%
lps_hasupper	6	3	1				2	√
lps_getupper	6	3	1				2	√
lps_setupper	14	8	2	6	1		3	√
lps_setlhs	14	8	2	6	1		3	66%
lps_setrhs	14	8	2	6	1		3	√
lps_setcontype	6	3	0				2	⊗
lps_contype	6	3	0				2	√
lps_vartype	6	3	0				2	⊗
lps_getclass	6	3	0				2	√
lps_setclass	6	3	0				2	√
lps_getlhs	6	3	1				2	⊗
lps_getrhs	6	3	1				2	⊗
lps_setvartype	6	3	0				2	⊗
lps_varstate	6	3	0				2	⊗
lps_setvarstate	6	3	0				2	⊗
lps_constate	6	3	0				2	⊗
lps_setconstate	6	3	0				2	⊗
lps_flags	6	3	0				2	⊗
lps_addflags	6	3	0				2	√
lps_setscale	6	3	1				2	⊗
lps_setpriority	6	3	0				2	√
lps_setvalue	6	3	1				2	⊗
lps_setstartval	6	3	1				2	√
lps_stat	6	2	2				1	⊗
lps_write	20	10	4	4	1		2	91%
lpfstrncpy	20	11	2	5	2			√
lps_makename	34	22	10	3	2		8	√
lps_transtable	35	24	12	8	2		5	√
lps_scale	32	26	15	8	3		1	95%
66 functions, total	975	672	198	181			195	76%
∅ per function	14.8	10.2	3.0	2.7			3.0	
∅ statements per	0.7		3.4	3.7			3.4	

ratmpswrite.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
write_data	15	5	7	2	1		2	√
write_vars	40	27	12	6	3		4	92%
mps_write	120	78	41	28	3		3	90%
3 functions, total	175	110	60	36			9	91%
∅ per function	58.3	36.7	20.0	12.0			3.0	
∅ statements per	0.6		1.8	3.1			11.0	

ratmstwrite.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
lps_mstfile	27	23	9	7	1		4	95%
1 functions, total	27	23	9	7			4	95%
∅ per function	27.0	23.0	9.0	7.0			4.0	
∅ statements per	0.9		2.6	3.3			4.6	

ratordwrite.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
lps_orderfile	34	28	11	9	1		4	89%
1 functions, total	34	28	11	9			4	89%
∅ per function	34.0	28.0	11.0	9.0			4.0	
∅ statements per	0.8		2.5	3.1			5.6	

ratpresolve.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
remove_fixed_var	39	25	16	7	3		3	69%
simple_rows	123	78	54	38	4	5	2	58%
handle_col_singleton	136	68	31	19	5	9	7	40%
simple_cols	92	55	27	21	4	5	3	40%
lps_presolve	26	15	3	8	1		2	81%
5 functions, total	416	241	131	93			17	51%
∅ per function	83.2	48.2	26.2	18.6			3.4	
∅ statements per	0.6		1.8	2.6			13.4	

rdefpar.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
rdef_new	16	13	4				4	√
rdef_free	10	5	3	2	1		1	⊗
rdef_is_valid	8	1	1	5				√
rdef_copy	7	4	1				1	√
rdef_set_param	20	9	2	4	1		2	54%
rdef_get_filename	5	2	1				1	√
rdef_get_template	5	2	1				1	√
rdef_get_comment	5	2	1				1	√
rdef_get_use	5	2	1				1	√
rdef_get_skip	5	2	1				1	√
rpar_new_skip	10	7	3				2	⊗
rpar_new_use	10	7	3				2	⊗
rpar_new_comment	10	7	3				2	√
rpar_free	6	3	3				1	⊗
rpar_is_valid	5	1	1	3				√
rpar_copy	11	8	3				3	⊗
16 functions, total	138	75	32	26			23	56%
∅ per function	8.6	4.7	2.0	1.6			1.4	
∅ statements per	0.5		2.3	2.9			3.1	

set4.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
set_init	13	9	7				2	√
set_exit	5	2	1				1	√
set_new_from_list	29	19	16	5	1		4	95%
set_free	4	1	1					√
set_is_valid	4	1	1	2				√
set_copy	4	1	1					√
set_lookup_idx	4	1	1					√
set_lookup	4	1	1					√
set_get_tuple_intern	4	1	1					√
set_get_tuple	10	7	3				3	√
set_iter_init_intern	4	1	1					√
set_iter_init	4	1	1					√
set_iter_next_intern	4	1	1					√
set_iter_next	9	6	3	2		2		√
set_iter_exit_intern	4	1	1					√
set_iter_exit	4	1	1					√
set_iter_reset_intern	4	1	1					√
set_get_dim	5	2	1				1	√
set_get_members	5	2	1				1	√
set_print	45	29	16	9	1		2	77%
set_union	45	29	23	7	2		5	90%
set_inter	32	21	15	5	2		4	95%
set_minus	35	22	15	5	2		5	91%
set_sdiff	48	30	23	8	2		5	90%
set_proj	48	37	28	6	1		6	94%
set_is_subseteq	30	22	12	6	1	4	2	91%
set_is_subset	8	5	3	2		2	2	√
set_is_equal	8	5	3	2		2	2	√
counter_inc	13	8	1	3	1	2		√
set_subsets_list	51	44	21	6	2		7	√
30 functions, total	487	311	204	84			52	93%
∅ per function	16.2	10.4	6.8	2.8			1.7	
∅ statements per	0.6		1.5	3.7			5.9	

setempty.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
set_empty_is_valid	7	1	1	4				√
set_empty_iter_is_valid	4	1	1	2				√
set_empty_new	13	10	3				2	√
set_empty_copy	6	3	0					√
set_empty_free	10	5	3	2	1		1	√
set_empty_lookup_idx	7	4	2				3	⊗
set_empty_get_tuple	11	8	4				6	⊗
iter_init	12	9	5	2			5	√
iter_next	6	3	2				2	√
iter_exit	6	3	3				1	√
iter_reset	4	1	1				1	⊗
set_empty_init	12	9	0					√
12 functions, total	98	57	25	18			21	76%
∅ per function	8.2	4.8	2.1	1.5			1.8	
∅ statements per	0.6		2.3	3.2			2.6	

setlist.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
set_list_is_valid	14	6	2	9		2		83%
set_list_iter_is_valid	9	1	1	6				√
lookup_elem_idx	12	10	4	4		3	2	√
set_list_new	19	16	5	3			4	√
set_list_add_elem	27	17	8	5	2		5	√
set_list_new_from_elems	14	11	6	2			3	√
set_list_new_from_tuples	19	14	8	2	1		4	√
set_list_new_from_entrie	19	14	9	2	1		4	√
set_list_copy	6	3	0					√
set_list_free	16	12	6	4	1		1	√
set_list_lookup_idx	8	5	5				4	√
set_list_get_tuple	10	7	5				6	√
set_list_iter_init	40	24	9	6	3		6	92%
set_list_iter_next	13	10	6	2		2	5	√
set_list_iter_exit	6	3	3				1	√
set_list_iter_reset	5	2	1				1	√
set_list_init	12	9	0					√
set_list_get_elem	7	4	1				3	√
18 functions, total	256	168	79	52			49	98%
∅ per function	14.2	9.3	4.4	2.9			2.7	
∅ statements per	0.7		2.1	3.2			3.4	

setmulti.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
set_multi_is_valid	9	1	1	6				√
set_multi_iter_is_valid	10	1	1	11				√
subset_cmp	14	10	0	3	1	2		90%
order_cmp	13	10	0				3	√
set_multi_new_from_list	91	67	29	16	3		10	√
set_multi_copy	10	8	2	2			1	√
set_multi_free	18	15	8	4	1		1	√
subset_idx_cmp	17	13	0	3	1	2	3	√
order_idx_cmp	13	10	0				7	√
set_multi_lookup_idx	35	26	8	4	2	3	8	88%
set_multi_get_tuple	16	10	6	2	1		6	√
set_multi_iter_init	120	79	14	18	4		19	92%
set_multi_iter_next	19	13	7	3	1	2	5	√
set_multi_iter_exit	8	5	4	2			1	√
set_multi_iter_reset	5	2	1				1	⊗
set_multi_init	12	9	0					√
16 functions, total	410	279	81	78			65	95%
∅ per function	25.6	17.4	5.1	4.9			4.1	
∅ statements per	0.7		3.4	3.6			4.2	

setprod.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
set_prod_is_valid	9	1	3	6				√
set_prod_iter_is_valid	7	1	1	4				√
set_prod_new	21	18	8	3		2	6	94%
set_prod_copy	8	5	2					√
set_prod_free	12	7	5	2	1		1	√

(continued on next page)

setprod.c *(cont.)*	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
set_prod_lookup_idx	17	14	5	3		3	4	86%
set_prod_get_tuple	17	14	5				6	√
set_prod_iter_init	19	15	9	3			7	√
get_both_parts	15	11	5	4	1	3	2	90%
set_prod_iter_next	45	32	11	6	2	3	6	√
set_prod_iter_exit	14	12	8	3			2	91%
set_prod_iter_reset	7	4	4				2	⊗
set_prod_init	12	9	0					√
13 functions, total	203	143	66	38			36	93%
∅ per function	15.6	11.0	5.1	2.9			2.8	
∅ statements per	0.7		2.2	3.8			3.9	

setpseudo.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
set_pseudo_is_valid	8	1	1	5				√
set_pseudo_iter_is_valid	4	1	1	2				√
set_pseudo_new	13	10	3				2	√
set_pseudo_copy	6	3	0					√
set_pseudo_free	10	5	3	2	1		1	√
set_pseudo_lookup_idx	10	7	4	2		2	4	87%
set_pseudo_get_tuple	8	5	3				5	⊗
iter_init	14	11	6	3			6	√
iter_next	9	6	2	2		2	2	√
iter_exit	6	3	3				1	√
iter_reset	5	2	1				1	⊗
set_pseudo_init	12	9	0					√
12 functions, total	105	63	27	22			22	86%
∅ per function	8.8	5.2	2.2	1.8			1.8	
∅ statements per	0.6		2.3	2.9			2.7	

setrange.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
set_range_is_valid	7	1	1	4				√
set_range_iter_is_valid	7	1	1	5				√
set_range_new	16	13	3				2	√
set_range_copy	6	3	0					√
set_range_free	10	5	3	2	1		1	√
idx_to_val	5	1	0					√
val_to_idx	5	1	0					√
set_range_lookup_idx	39	23	11	10	1	5	5	83%
set_range_get_tuple	15	12	8				6	√
set_range_iter_init	48	30	12	8	3		6	75%
set_range_iter_next	18	15	9	2		2	5	√
set_range_iter_exit	6	3	3				1	√
set_range_iter_reset	5	2	1				1	√
set_range_init	12	9	0					√
14 functions, total	199	119	52	39			27	91%
∅ per function	14.2	8.5	3.7	2.8			1.9	
∅ statements per	0.6		2.3	3.1			4.2	

source.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
show_source	31	24	3	5	1		6	√
1 functions, total	31	24	3	5			6	100%
∅ per function	31.0	24.0	3.0	5.0			6.0	
∅ statements per	0.8		8.0	4.8			3.4	

stmt.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
stmt_new	16	14	5				5	√
stmt_free	10	7	6	2			1	√
stmt_is_valid	8	1	1	5				√
stmt_get_filename	5	2	1				1	√
stmt_get_lineno	5	2	1				1	√
stmt_get_text	5	2	1				1	√
stmt_parse	9	6	5	2			1	83%
stmt_execute	13	7	6	3	1		1	57%
stmt_print	16	4	2		1		2	⊗
9 functions, total	87	45	28	17			13	82%
∅ per function	9.7	5.0	3.1	1.9			1.4	
∅ statements per	0.5		1.6	2.6			3.2	

strstore.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
str_new	11	8	2				3	√
str_init	3	0	0					√
str_exit	12	8	2	2	1			√
str_hash	8	6	1	2				√
4 functions, total	34	22	5	6			3	100%
∅ per function	8.5	5.5	1.2	1.5			0.8	
∅ statements per	0.6		4.4	3.7			5.5	

symbol.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
symbol_new	25	22	8	2			7	√
symbol_exit	21	18	8	4	1		1	√
symbol_is_valid	4	1	1	2				√
symbol_lookup	9	7	1	3			1	√
symbol_has_entry	7	3	4	3			2	⊗
symbol_lookup_entry	10	7	4	4			2	√
symbol_add_entry	36	23	13	6	1		7	√
symbol_get_dim	5	2	2				1	⊗
symbol_get_iset	5	2	1				1	√
symbol_get_name	5	2	1				1	√
symbol_get_type	5	2	1				1	√
symbol_get_numb	7	4	2				3	⊗
symbol_get_strg	7	4	2				3	⊗
symbol_get_set	7	4	2				3	⊗
symbol_get_var	7	4	2				3	⊗
symbol_print	18	14	10	2	1		1	⊗
symbol_print_all	7	5	1	2			1	⊗
17 functions, total	185	124	63	36			38	66%
∅ per function	10.9	7.3	3.7	2.1			2.2	
∅ statements per	0.7		2.0	3.4			3.2	

term.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
term_new	14	11	6				3	√
term_add_elem	20	14	8	2	1		7	√
term_free	12	10	7	2			1	√
term_copy	17	13	7	2	1		3	√
term_append_term	11	9	6	2			3	√
term_add_term	25	20	9	3	1		4	√
term_sub_term	26	21	10	3	1		4	√
term_add_constant	7	4	4				2	√
term_sub_constant	7	4	4				2	√
term_mul_coeff	19	13	8	4	1		2	√
term_get_constant	5	2	1				1	√
term_negate	6	3	4				1	⊗
term_to_nzo	15	11	8	2	1		4	√
term_to_objective	14	10	8	2	1		3	√
term_get_elements	5	2	1				1	√
term_get_lower_bound	28	21	15	4	1		1	95%
term_get_upper_bound	28	21	15	4	1		1	√
term_is_all_integer	12	8	2	4	1	2		√
18 functions, total	271	197	123	40			43	88%
∅ per function	15.1	10.9	6.8	2.2			2.4	
∅ statements per	0.7		1.6	4.9			4.5	

tuple.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
tuple_new	17	13	4	2			4	√
tuple_free	16	12	5	4	1		2	√
tuple_is_valid	4	1	1	3				√
tuple_copy	7	4	1				1	√
tuple_cmp	27	19	8	7	2	3	3	68%
tuple_get_dim	5	2	1				1	√
tuple_set_elem	9	6	1				5	√
tuple_get_elem	9	5	1				4	√
tuple_combine	13	12	5	3			2	⊗
tuple_print	12	8	5	3	1		1	√
tuple_hash	8	6	2	2				√
tuple_tostr	31	24	11	5	2		5	87%
12 functions, total	158	112	45	33			28	82%
∅ per function	13.2	9.3	3.8	2.8			2.3	
∅ statements per	0.7		2.5	3.4			3.9	

vinst.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
create_new_constraint	17	14	11				5	√
create_new_var_entry	22	19	19				5	√
check_how_fixed	34	21	12	13	1			⊗
check_if_fixed	64	38	35	21	3		5	42%
handle_vbool_cmp	212	156	189	18	2		4	89%
i_vbool_ne	5	2	2					√
i_vbool_eq	5	2	2					√
i_vbool_lt	5	2	2					√
i_vbool_le	5	2	2					√
i_vbool_gt	5	2	2					√

(continued on next page)

vinst.c *(cont.)*	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
i_vbool_ge	5	2	2					√
i_vbool_and	45	36	34				3	√
i_vbool_or	45	36	34				3	√
i_vbool_xor	52	42	40				3	√
i_vbool_not	34	27	22				2	√
gen_cond_constraint	50	33	25	13	2		1	91%
handle_vif_then_else	25	17	13	8	1		1	√
i_vif_else	24	19	15				1	√
i_vif	19	15	10				1	√
i_vabs	114	90	98	10	1		2	95%
20 functions, total	787	575	569	97			36	87%
∅ per function	39.4	28.8	28.4	4.8			1.8	
∅ statements per	0.7		1.0	5.9			15.5	

xlpglue.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
xlp_alloc	5	2	1				1	√
xlp_scale	4	1	1					√
xlp_write	5	2	1				1	√
xlp_transtable	5	2	1				1	√
xlp_orderfile	5	2	1				1	√
xlp_mstfile	5	2	1				1	√
xlp_free	5	2	1					√
xlp_stat	4	1	1					⊗
xlp_conname_exists	5	2	1				1	√
xlp_addcon	37	25	13	5	1		5	96%
xlp_addvar	33	24	21	4	1		5	96%
xlp_getclass	5	2	1				1	√
xlp_getlower	19	13	9	2	1		1	92%
xlp_getupper	19	13	9	2	1		1	√
xlp_objname	5	2	1				1	√
xlp_setdir	4	1	1	2				√
xlp_addtonzo	26	19	12	3	1		3	√
xlp_addtocost	15	12	8				2	√
xlp_presolve	21	10	4	5	1			33%
19 functions, total	227	137	88	35			25	91%
∅ per function	11.9	7.2	4.6	1.8			1.3	
∅ statements per	0.6		1.6	3.9			5.3	

zimpl.c	Lines	Stmt.	Calls	CC	Dp.	Ex.	As.	Cover
add_extention	11	8	6				4	√
strip_path	8	5	2	2			2	√
strip_extension	18	14	4	8	1		2	85%
check_write_ok	8	3	3	2	1			50%
is_valid_identifier	11	6	2	5		2	1	83%
add_parameter	44	33	23	6	1	2	1	80%
main	247	173	77	49	3		1	70%
7 functions, total	347	242	117	73			11	74%
∅ per function	49.6	34.6	16.7	10.4			1.6	
∅ statements per	0.7		2.1	3.3			20.2	

Appendix C

Zimpl Programs

C.1 Facility location model with discrete link capacities

For the description of the model see Section . on page .

```
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
# *                                                                 *
# *   File ....: gwin.zpl                                           *
# *   Name ....: Three layer facility location                      *
# *   Author ..: Thorsten Koch                                      *
# *                                                                 *
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
set K     := { "T2", "T3", "T4", "T5" };
set V     := { read "backbone.dat" as "<1s>" comment "#" };
set W     := V;
set U     := { read "nodes.dat" as "<1s>" comment "#" };
set AUV   := { read "auv.dat" as "<1s,2s>" comment "#" };
set AVW   := { <u,v> in AUV with <u> in V };
set AVWxK := AVW * K;

param capa[K]    := <"T2"> 34,  <"T3"> 155, <"T4"> 622, <"T5"> 2400;
param cost[K]    := <"T2"> 2.88,<"T3"> 6.24,<"T4"> 8.64,<"T5"> 10.56;
param cxuv[AUV]  := read "auv.dat"      as "<1s,2s> 3n" comment "#";
param dist[AUV]  := read "auv.dat"      as "<1s,2s> 4n" comment "#";
param cy[V]      := read "backbone.dat" as "<1s> 2n"    comment "#" ;
param demand[U]  := read "nodes.dat"    as "<1s> 4n"    comment "#";
param min_dist   := 50;

var xuv[AUV]   binary;
var xvw[AVWxK] binary;
var yv [V]     binary;
var yw [W]     binary;

minimize cost:
   sum <u,v>   in AUV   : cxuv[u,v] * xuv[u,v]
 + sum <v,w,k> in AVWxK : dist[v,w] * cost[k] * xvw[v,w,k]
 + sum <v>     in V     : cy[v] * yv[v]
 + sum <w>     in W     : cy[w] * yw[w];
```

```
# All demand nodes have to be connected.
subto c1: forall <u> in U do sum <u,v> in AUV : xuv[u,v] == 1;

# Demand nodes can only be connected to active backbone nodes.
subto c2: forall <u,v> in AUV do xuv[u,v] <= yv[v];

# If backbone nodes are active they have to be connected to a core
# node with a link from one of the possible capacities.
subto c3: forall <v> in V do
   sum <v,w,k> in AVWxK : xvw[v,w,k] == yv[v];

# Backbone nodes can only be connected to active core nodes.
subto c4: forall <v,w> in AVW do sum <k> in K : xvw[v,w,k] <= yw[w];

# Backbone nodes that are not identical to a core node have
# to be at least min_dist apart.
subto c5:
   forall <v,w,k> in AVWxK with dist[v,w] < min_dist and v != w do
      xvw[v,w,k] == 0;

# Each core node is connected to exactly three backbone nodes.
subto c6: forall <w> in W do
   sum <v,w,k> in AVWxK : xvw[v,w,k] == 3 * yw[w];

# We have ten core nodes.
subto c7: sum <w> in W : yw[w] == 10;

# A core node has to be also a backbone node.
subto c8: forall <w> in W : yw[w] <= yv[w];

# The capacity of the link from the backbone to the core node
# must have sufficient capacity.
subto c9: forall <v> in V do
   sum <u,v> in AUV : demand[u] * xuv[u,v] <=
   sum <v,w,k> in AVWxK : capa[k] * xvw[v,w,k];
```

C.2 Facility location model with configurations

For the description of the model see Section on page .

```
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
# *                                                                 *
# *   File....: facility.zpl                                        *
# *   Name....: Facility location with configurations               *
# *   Author..: Thorsten Koch                                       *
# *                                                                 *
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
set S   := { read "cfg.dat" as "<1s>" comment "#" };
set U   := { read "bsc.dat" as "<1s>" comment "#" };
set V   := { read "msc.dat" as "<1s>" comment "#" };
set A   := { read "auv.dat" as "<1s,2s>" comment "#" };
set VxS := V * S;
```

```
var x[A]   binary;
var z[VxS] binary;

param demand[U] := read "bsc.dat" as "<1s> 2n" comment "#" ;
param kappa [S] := read "cfg.dat" as "<1s> 2n" comment "#" ;
param cx    [A] := read "auv.dat" as "<1s,2s> 3n" comment "#" ;
param cz    [S] := read "cfg.dat" as "<1s> 3n" comment "#" ;

minimize cost:
   sum <u,v> in A   : cx[u,v] * x[u,v]
 + sum <v,s> in VxS : cz[s]   * z[v,s];

# Each BSC has to be connected to exactly one MSC.
subto c1: forall <u> in U do sum <u,v> in A : x[u,v] == 1;

# Each MSC has exactly one confguration.
subto c2: forall <v> in V do sum <v,s> in VxS : z[v,s] == 1;

# The configurations at the MSC need to have enough
# capacity to meet the demands of all connected layer-1 nodes.
subto c3: forall <v> in V do
   sum <u,v> in A   : demand[u] * x[u,v]
 - sum <v,s> in VxS : kappa[s]  * z[v,s] <= 0;
```

C.3 UMTS site selection

For the description of the model see Section . . on page .

```
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
# *                                                               *
# *   File....: siteselect.zpl                                    *
# *   Name....: UMTS Site Selection                               *
# *   Author..: Thorsten Koch                                     *
# *                                                               *
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
set SP  := { read "site_cover.dat" as "<1n,2n>" comment "#" };
set S   := proj(SP, <1>);
set P   := proj(SP, <2>);
set SxS := { read "site_inter.dat" as "<1n,2n>" comment "#" };

param cz[SxS] := read "site_inter.dat" as "<1n,2n> 3n" comment "#";

var x[S]   binary;
var z[SxS] binary;

minimize sites:
   sum <s>     in S   : 100 * x[s]
 + sum <s1,s2> in SxS : cz[s1,s2] * z[s1,s2];

# Each pixel has to be covered.
subto c1: forall <p> in P do sum <s,p> in SP : x[s] >= 1;

# Mark sites interfering with each other.
subto c2: forall <i,j> in SxS do z[i,j] - x[i] - x[j] >= -1;
```

C.4 UMTS azimuth setting

For the description of the model see Section . . on page .

```
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
# *                                                                 *
# *   File....: azimuth.zpl                                        *
# *   Name....: UMTS Azimuth Selection                             *
# *   Author..: Thorsten Koch                                      *
# *                                                                 *
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
set A := { read "arcs.dat"  as "<1n,2n>"         comment "#" };
set C := { read "inter.dat" as "<1n,2n,3n,4n>" comment "#" };
set F := { read "angle.dat" as "<1n,2n,3n,4n>" comment "#" };
set S := proj(A, <1>);

param cost [A] := read "arcs.dat"  as "<1n,2n> 3n" comment "#";
param cells[S] := read "cells.dat" as "<1n> 2n"    comment "#" ;
param angle[F] := read "angle.dat"
                         as "<1n,2n,3n,4n> 5n" comment "#" ;
var x[A] binary;
var y[F] binary;
var z[C] binary;

minimize cost:
   sum <i,j> in A       : (cost[i,j] / 10)^2 * x[i,j]
 + sum <i,j,m,n> in F : (angle[i,j,m,n] / 10)^2 * y[i,j,m,n]
 + sum <i,j,m,n> in C : 200 * z[i,j,m,n];

# Each site s has cells[s] cells.
subto c1: forall <s> in S do sum <s,i> in A : x[s,i] == cells[s];

# Only one direction allowed, i to j or j to i or neither.
subto c2: forall <i,j> in A with i < j do x[i,j] + x[j,i] <= 1;

# Mark interfering angles.
subto c3: forall <i,j,m,n> in F do
   y[i,j,m,n] - x[i,j] - x[m,n] >= -1;

# Mark beams that cross each other
subto c4: forall <i,j,m,n> in C do
   z[i,j,m,n] - x[i,j] - x[m,n] >= -1;
```

C.5 UMTS snapshot model

For the description of the model see Section . . on page .

```
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
# *                                                                 *
# *                    This file is part of the tool set            *
# *     schnappfisch --- UMTS Snapshot Generator and Evaluator     *
# *                                                                 *
# *   Copyright (C) 2002 Thorsten Koch                              *
# *                      2002 Atesio GmbH                           *
```

```
# *                   2002 Konrad-Zuse-Zentrum                          *
# *                        fuer Informationstechnik Berlin              *
# *                   2002 Technische äUniversitt Darmstadt             *
# *                                                                     *
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
set S   := { read "sites.dat"         as "<1s>" comment "#" };
set I   := { read "installations.dat" as "<1s>" comment "#" };
set M   := { read "mobiles.dat"       as "<1s>" comment "#" };
set C   := { read "userclass.dat"     as "<1s>" comment "#" };
set IM := I * M;

param min_inst       [S] := read "sites.dat" as "<1s> 2n";
param max_inst       [S] := read "sites.dat" as "<1s> 3n";
param atten_dl      [IM] := read "attenuation.dat" as "<1s,2s> 3n";
param atten_ul      [IM] := read "attenuation.dat" as "<1s,2s> 4n";
param orthogo       [IM] := read "attenuation.dat" as "<1s,2s> 5n";
param pmin_pilot     [I] := read "installations.dat" as "<1s> 4n";
param pmax_pilot     [I] := read "installations.dat" as "<1s> 5n";
param pmin_link      [I] := read "installations.dat" as "<1s> 6n";
param pmax_link      [I] := read "installations.dat" as "<1s> 7n";
param pmax_down      [I] := read "installations.dat" as "<1s> 8n";
param code_budget    [I] := read "installations.dat" as "<1s> 9n";
param at_site        [I] := read "installations.dat" as "<1s> 2s";
param noise_i        [I] := read "installations.dat" as "<1s> 3n";
param cchf_i         [I] := read "installations.dat" as "<1s> 10n";
param max_nrise_i    [I] := read "installations.dat" as "<1s> 11n";
param snap           [M] := read "mobiles.dat" as "<1s>  2n";
param userclass      [M] := read "mobiles.dat" as "<1s>  3s";
param noise_m        [M] := read "mobiles.dat" as "<1s>  4n";
param pmin_up        [M] := read "mobiles.dat" as "<1s>  5n";
param pmax_up        [M] := read "mobiles.dat" as "<1s>  6n";
param activity_ul    [M] := read "mobiles.dat" as "<1s>  7n";
param activity_dl    [M] := read "mobiles.dat" as "<1s>  8n";
param code_length    [M] := read "mobiles.dat" as "<1s>  9n";
param min_rscp_pilot [M] := read "mobiles.dat" as "<1s> 10n";
param cir_pilot      [M] := read "mobiles.dat" as "<1s> 11n";
param cir_up         [M] := read "mobiles.dat" as "<1s> 12n";
param cir_down       [M] := read "mobiles.dat" as "<1s> 13n";
param cover_target   [C] := read "userclass.dat" as "<1s> 2n";
param users          [C] := read "userclass.dat" as "<1s> 3n";

set D   := { 0 to max <m> in M : snap[m] };
set ID := I * D;

param pi_scale [<i,d> in ID] :=
   max <m> in M with snap[m] == d : atten_ul[i,m] * activity_ul[m];

param serviceable [<i,m> in IM] :=
   if (     atten_dl[i,m] * pmax_pilot[i] >= cir_pilot[m] * noise_m[m]
        and atten_dl[i,m] * pmax_pilot[i] >= min_rscp_pilot[m])
   then 1 else 0 end;

param demand [<d,c> in D * C] :=
   sum <m> in M with snap[m] == d and userclass[m] == c
```

```
      and (max <i> in I : serviceable[i,m]) == 1 : 1;

var s[S]              binary;
var z[I]              binary;
var u[M]              binary;
var x[<i,m> in IM]    integer <= serviceable[i,m];

var pu [<m> in M]     real <= pmax_up[m];
var pd [<i,m> in IM]  real <= pmax_link[i];
var pp [<i> in I]     real <= pmax_pilot[i];
var pt [<i,d> in ID]  real <= pmax_down[i];
var pi [<i,d> in ID]  real;
var bod               integer <= card(M);

minimize cost:
   sum <k>   in S  : 1000  * s[k]
 + sum <i>   in I  :  100  * z[i]
 + sum <m>   in M  : 1000  * u[m]
 +                    500  * bod
 + sum <i,m> in IM :   -1  * x[i,m]
 + sum <m>   in M  :  0.1  * pu[m]
 + sum <i,m> in IM :  0.2  * pd[i,m]
 + sum <i>   in I  :  0.05 * pp[i]
 + sum <i,d> in ID :  0.1  * pt[i,d];

# Only active sites can have installations.
subto c1: forall <i> in I do z[i] - s[at_site[i]] <= 0;

# There may be sites which have a minimum number of cells.
subto c2a: forall <k> in S do
   sum <i> in I with at_site[i] == k : z[i] >= min_inst[k];

# The number of installations per site is limited.
subto c2b: forall <k> in S do
   sum <i> in I with at_site[i] == k : z[i] <= max_inst[k];

# Only active installations may serve mobiles.
subto c3: forall <i,m> in IM with serviceable[i,m] == 1 do
   x[i,m] - z[i] <= 0;

# Count the unserved mobiles.
subto c4: forall <m> in M do
   u[m] + sum <i> in I with serviceable[i,m] == 1 : x[i,m] == 1;

# Count blocked or dropped mobiles per snapshot.
subto c4a: forall <d> in D do
   sum <m> in M with snap[m] == d : u[m] - bod <= 0;

# Only a limited number of codes is available.
subto c5: forall <i,d> in ID do
   sum <m> in M with snap[m] == d and serviceable[i,m] == 1 :
      1 / code_length[m] * x[i,m] - code_budget[i] * z[i] <= 0;

# Every served mobile emits a minimum power.
```

```
subto c6a: forall <m> in M do
   pu[m] + pmin_up[m] * u[m] >= pmin_up[m];

# Max power also is limited.
subto c6b: forall <m> in M do
   pu[m] + pmax_up[m] * u[m] <= pmax_up[m];

# Received power (interference) at base station.
subto c7a: forall <i,d> in ID do
   sum <m> in M with snap[m] == d and serviceable[i,m] == 1 :
      atten_ul[i,m] * activity_ul[m] * pu[m]
     - pi_scale[i,d] * pi[i,d] == 0, scale;

# Limit noise rise for active installations.
subto c7b: forall <i,d> in ID do
   + pi_scale[i,d] * pi[i,d]
   + sum <m> in M with snap[m] == d and serviceable[i,m] == 1 :
       atten_ul[i,m] * activity_ul[m] * pmax_up[m] * z[i]
   <= noise_i[i] * (max_nrise_i[i] - 1)
   + sum <m> in M with snap[m] == d and serviceable[i,m] == 1 :
        atten_ul[i,m] * activity_ul[m] * pmax_up[m], scale;

# CIR uplink.
subto c8: forall <i> in I do
   forall <d> in D do
      forall <m> in M with snap[m] == d
                       and serviceable[i,m] == 1 do
      - (atten_ul[i,m] / cir_up[m]) * pu[m]
      + pi_scale[i,d] * pi[i,d]
      - atten_ul[i,m] * activity_ul[m] * pu[m]
      + noise_i[i] * x[i,m]
      + sum <n> in M with n != m and snap[n] == d :
         atten_ul[i,n] * activity_ul[n] * pmax_up[n] * x[i,m]
      <= sum <n> in M with n != m and snap[n] == d:
         atten_ul[i,n] * activity_ul[n] * pmax_up[n], scale;

# Min and max downlink power must be used if mobile is served.
subto c9a: forall <i,m> in IM with serviceable[i,m] == 1 do
    pmin_link[i] * x[i,m] - pd[i,m] <= 0;

subto c9b: forall <i,m> in IM with serviceable[i,m] == 1 do
    pmax_link[i] * x[i,m] - pd[i,m] >= 0;

# Min and max pilot power must be used if installation is used.

subto c10a: forall <i> in I do pmin_pilot[i] * z[i] - pp[i] <= 0;
subto c10b: forall <i> in I do pmax_pilot[i] * z[i] - pp[i] >= 0;

# Limit the maximum power per installation.
subto c11a: forall <i,d> in ID do
   (1.0 + cchf_i[i]) * pp[i]
   + sum <m> in M with snap[m] == d and serviceable[i,m] == 1 :
       activity_dl[m] * pd[i,m] - pt[i,d] == 0;
```

```
subto c11b: forall <i,d> in ID do
   pt[i,d] - pmax_down[i] * z[i] <= 0;

# CIR downlink.
subto c12: forall <i> in I do
   forall <d> in D do
      forall <m> in M with snap[m] == d
                  and serviceable[i,m] == 1 do
      - (atten_dl[i,m] / cir_down[m]) * pd[i,m]
      + orthogo[i,m] * atten_dl[i,m] * pt[i,d]
      - orthogo[i,m] * atten_dl[i,m] * activity_dl[m] * pd[i,m]
      + sum <j> in I with j != i : atten_dl[j,m] * pt[j,d]
      + orthogo[i,m] * atten_dl[i,m] * pmax_down[i] * x[i,m]
      + sum <j> in I with j != i :
           atten_dl[j,m] * pmax_down[j] * x[i,m]
           + noise_m[m] * x[i,m]
      <= orthogo[i,m] * atten_dl[i,m] * pmax_down[i]
      + sum <j> in I with j != i :
           atten_dl[j,m] * pmax_down[j], scale;

# CIR Pilot.
subto c13: forall <i> in I do
   forall <d> in D do
      forall <m> in M with snap[m] == d
                      and serviceable[i,m] == 1 do
      - (atten_dl[i,m] / cir_pilot[m]) * pp[i]
      + sum <j> in I : atten_dl[j,m] * pt[j,d]
      - atten_dl[i,m] * pp[i]
      + sum <j> in I : atten_dl[j,m] * pmax_down[j] * x[i,m]
      + noise_m[m] * x[i,m]
      <= sum <j> in I : atten_dl[j,m] * pmax_down[j], scale;

# Minimum Pilot RSCP.
subto c14: forall <i,m> in IM with serviceable[i,m] == 1 do
   atten_dl[i,m] * pp[i] - min_rscp_pilot[m] * x[i,m] >= 0, scale;

# Coverage requirement.
subto c15: forall <d,c> in D * C do
   sum <m> in M with snap[m] == d and userclass[m] == c
      and (max <i> in I : serviceable[i,m]) == 1 : u[m]
   <= floor((1 - cover_target[c]) * demand[d,c]);

# MIR-cut based on uplink CIR target inequality.
subto c8s: forall <i,m> in IM with serviceable[i,m] == 1 do
  + 1 * x[i,m] - atten_ul[i,m]
  * (1 / cir_up[m] + activity_ul[m]) / noise_i[i] * pu[m]
  <= 0, scale;

# MIR-cut based on downlink CIR target inequality.
subto c12s: forall <i,m> in IM with serviceable[i,m] == 1 do
  + 1 * x[i,m] - atten_dl[i,m]
  * (1 / cir_down[m] + orthogo[i,m] * activity_dl[m])
  / noise_m[m] * pd[i,m] <= 0, scale;
```

```
# Heuristic best server.
subto h1:
   forall <i,m> in IM with serviceable[i,m] == 1
      and atten_ul[i,m] * pmax_up[m]
       <= cir_up[m] * max_nrise_i[i] * noise_i[i] do
      x[i,m] <= 0;

# Dominance criterion.
subto d1: forall <m> in M do
      forall <n> in M with n != m and snap[n] == snap[m]
         and activity_ul[n] >=  activity_ul[m]
         and activity_dl[n] >=  activity_dl[m]
         and cir_up[n] >= cir_up[m] and cir_down[n] >= cir_down[m]
         and pmax_up[n] <= pmax_up[m] do
         forall <i> in I with serviceable[i,n] == 1
            and atten_ul[i,m] * pmin_up[m]
            <= cir_up[m] * (atten_ul[i,n] * activity_ul[n]
               * pmin_up[n] + noise_i[i])
         and atten_dl[i,n] < atten_dl[i,m]
         and atten_ul[i,n] < atten_ul[i,m] do
            x[i,n] - sum <k> in I with serviceable[k,m] == 1
            and atten_dl[k,m] >= atten_dl[i,m]
            and atten_ul[k,m] >= atten_ul[i,m]
            do x[k,m] <= 0;
```

C.6 Steiner tree packing

For the description of the model see Section . . on page .

```
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
# *                                                                 *
# *   File....: stp3d.zpl                                           *
# *   Name....: Steiner Tree Packing                                *
# *   Author..: Thorsten Koch                                       *
# *                                                                 *
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
set   Parameter := { "nodes", "nets" };
param parameter[Parameter] :=
         read "param.dat" as "<1s> 2n" comment "#";

set L := { 1 .. parameter["nets"] };                    # Nets
set V := { 1 .. parameter["nodes"] };                   # Nodes
set S := { read "terms.dat" as "<1n>"    comment "#" }; # Terms+Roots
set R := { read "roots.dat" as "<1n>"    comment "#" }; # Roots
set A := { read "arcs.dat"  as "<1n,2n>" comment "#" };
set T := S - R;                                         # only Terms
set N := V - S;                                         # Normal

param innet[S] := read "terms.dat" as "<1n> 2n"    comment "#";
param cost [A] := read "arcs.dat" as "<1n,2n> 3n" comment "#";

var y[A * T] binary;
var x[A * L] binary;
```

```
minimize obj: sum <i,j,k> in A * L : cost[i,j] * x[i,j,k];

# For all roots flow out.
subto c1a: forall <t> in T do
   forall <r> in R do
      sum <r,j> in A : y[r,j,t] ==
         if innet[r] == innet[t] then 1 else 0 end;

# For all roots flow in.
subto c1b: forall <t> in T do
   forall <r> in R do sum <j,r> in A : y[j,r,t] == 0;

# For all terminals flow out.
subto c2a: forall <t> in T do
   sum <t,j> in A : y[t,j,t] == 0;

# For all terminals in their own net: one flow in.
subto c2b: forall <t> in T do
  sum <j,t> in A : y[j,t,t] == 1;

# For all terminals in the same net: in equals out.
subto c2c: forall <t> in T do
   forall <s> in T with s != t and innet[s] == innet[t] do
      sum <j,s> in A : (y[j,s,t] - y[s,j,t]) == 0;

# For all terminals in a different net: zero flow.
subto c2d: forall <t> in T do
   forall <s> in T with innet[s] != innet[t] do
      sum <j,s> in A : (y[j,s,t] + y[s,j,t]) == 0;

# For normal nodes: flow balance.
subto c3: forall <t> in T do
   forall <n> in N do sum <n,i> in A : (y[n,i,t] - y[i,n,t]) == 0;

# Bind x to y.
subto c4: forall <t> in T do
   forall <i,j> in A do y[i,j,t] <= x[i,j,innet[t]];

# Only one x can be active per arc.
subto c5: forall <i,j> in A with i < j do
   sum <k> in L : (x[i,j,k] + x[j,i,k]) <= 1;

# For a normal node only one incomming arc can be active.
subto c6: forall <n> in N do
   sum <j,n,k> in A * L : x[j,n,k] <= 1;
```

Appendix D

Steiner Tree Packing Instances

The first three numbers of each line describe the x, y, and z-position of a node. If there is a fourth number, the node is a terminal and the number designates the network the node belongs to. If there is no fourth number, the node is blocked. If the z-position is omitted, the node is blocked in all layers.

sb3-30-26d

1	1	1	13	1	21	1	23	8	1	1	28	17	1	1	22	30	1	1	11	31	17	1	21
1	2	1	1	1	23	1	1	8	31	1	16	17	31	1	14	30	31	1	13	31	20	1	15
1	4	1	13	1	24	1	3	9	1	1	26	19	31	1	25	31	1	1	18	31	21	1	6
1	4	1	28	1	26	1	12	9	31	1	7	20	1	1	23	31	3	1	25	31	22	1	8
1	7	1	27	1	27	1	21	10	1	1	2	20	31	1	7	31	4	1	19	31	23	1	19
1	8	1	29	1	28	1	14	11	1	1	29	21	1	1	12	31	5	1	4	31	24	1	19
1	9	1	5	1	29	1	10	11	31	1	5	21	31	1	24	31	7	1	10	31	25	1	3
1	10	1	8	2	1	1	2	12	1	1	26	22	1	1	17	31	8	1	16	31	27	1	14
1	12	1	18	2	31	1	18	12	31	1	9	22	31	1	23	31	9	1	22	31	28	1	12
1	13	1	15	3	31	1	29	13	1	1	6	23	1	1	9	31	10	1	11	31	29	1	20
1	14	1	11	4	1	1	2	13	31	1	4	23	31	1	16	31	11	1	8	31	30	1	7
1	15	1	17	4	31	1	27	14	31	1	22	24	1	1	6	31	12	1	26	31	31	1	10
1	16	1	24	6	1	1	27	15	1	1	20	26	1	1	24	31	13	1	3				
1	17	1	5	6	31	1	25	15	31	1	9	28	31	1	1	31	14	1	15				
1	20	1	28	7	31	1	17	16	31	1	20	29	1	1	21	31	16	1	4				

sb11-20-7

1	1	1	6	1	14	1	5	4	21	1	5	11	1	1	4	17	21	1	3	21	8	1	2
1	2	1	3	1	15	1	2	5	1	1	5	11	21	1	2	18	1	1	2	21	10	1	6
1	3	1	4	1	16	1	1	5	21	1	5	12	1	1	1	18	21	1	6	21	11	1	6
1	4	1	2	1	17	1	5	6	1	1	5	12	21	1	6	19	1	1	7	21	12	1	1
1	5	1	6	1	18	1	2	6	21	1	7	13	1	1	1	19	21	1	3	21	13	1	3
1	6	1	7	1	19	1	7	7	1	1	1	13	21	1	2	20	1	1	6	21	14	1	4
1	7	1	1	1	20	1	3	7	21	1	1	14	1	1	3	20	21	1	6	21	15	1	4
1	8	1	5	1	21	1	7	8	1	1	5	14	21	1	3	21	1	1	6	21	16	1	2
1	9	1	1	2	1	1	4	8	21	1	2	15	1	1	3	21	2	1	7	21	17	1	5
1	10	1	5	2	21	1	4	9	1	1	3	15	21	1	7	21	3	1	4	21	18	1	2
1	11	1	7	3	1	1	7	9	21	1	4	16	1	1	1	21	4	1	3	21	19	1	1
1	12	1	2	3	21	1	4	10	1	1	4	16	21	1	6	21	5	1	7	21	21	1	7
1	13	1	1	4	1	1	6	10	21	1	5	17	1	1	3	21	7	1	4				

dense-3

7	1	2	1	1	10	1	3	9	1	2	7	15	9	1	10	13	17	2	13	15	1	1	16
15	13	1	1	14	1	2	4	15	14	1	7	11	17	2	10	1	12	1	13	3	17	2	17
7	17	2	1	15	16	1	4	6	17	2	7	1	8	1	10	11	1	2	14	1	13	1	17
1	9	1	1	1	16	1	4	6	1	2	8	15	15	1	11	15	4	1	14	2	17	2	18
13	1	2	2	4	1	2	5	15	10	1	8	12	17	2	11	1	4	1	14	1	11	1	18
15	7	1	2	4	17	2	5	5	17	2	8	1	15	1	11	15	5	1	15	15	1	2	19
1	7	1	2	1	14	1	5	10	1	2	9	15	8	1	12	15	17	2	15	15	2	1	19
3	1	2	3	2	1	2	6	15	17	1	9	1	5	1	12	1	3	1	15	1	17	2	19
15	11	1	3	15	6	1	6	1	17	1	9	5	1	2	13	1	1	2	16	1	2	1	19
8	17	2	3	9	17	2	6	8	1	2	10	15	12	1	13	12	1	2	16				

taq-3

5	12	1	1	3	17	8	16	13	3	16	15	21	3
8	24	1	1	3	18	8	17	13	4	16	16	21	4
15	12	1	1	3	21	8	18	13	5	16	17	21	5
20	12	1	1	3	22	8	21	13	8	16	18	21	8
23	11	1	1	3	23	8	22	13	9	16	21	21	9
23	24	1	1	5	8	8	23	13	10	16	22	21	10
5	25	1	2	5	9	9	21	13	12	16	23	21	11
10	12	1	2	5	10	9	22	13	13	16	24	21	12
25	12	1	2	5	11	9	23	13	15	18	3	21	13
8	11	1	3	5	13	9	24	13	16	18	4	21	15
15	25	1	3	5	15	10	2	13	17	18	5	21	16
4	14	1	4	5	16	10	3	13	18	18	8	21	17
9	1	1	4	5	17	10	4	13	21	18	9	21	18
9	14	1	5	5	18	10	5	13	22	18	10	21	21
10	25	1	5	5	21	10	8	13	23	18	12	21	22
13	11	1	6	5	22	10	9	15	2	18	13	21	23
13	24	1	6	5	23	10	10	15	3	18	15	21	24
14	1	1	7	5	24	10	11	15	4	18	16	22	3
14	14	1	7	6	8	10	13	15	5	18	17	22	4
18	1	1	8	6	9	10	15	15	8	18	18	22	5
18	11	1	8	6	10	10	16	15	9	18	21	23	3
18	24	1	9	6	11	10	17	15	10	18	22	23	4
19	14	1	9	6	12	10	18	15	11	18	23	23	5
20	25	1	10	6	13	11	2	15	13	19	3	23	8
24	14	1	10	6	15	11	3	15	15	19	4	23	9
25	2	1	11	6	16	11	4	15	16	19	5	23	10
25	25	1	11	6	17	11	5	15	17	20	3	23	12
11	1	1	12	6	18	11	8	15	18	20	4	23	13
25	1	1	12	6	21	11	9	15	21	20	5	23	15
1	9	1	12	6	22	11	10	15	22	20	8	23	16
25	19	1	13	6	23	11	11	15	23	20	9	23	17
4	24	1	13	6	24	11	12	15	24	20	10	23	18
1	22	1	13	8	2	11	13	16	2	20	11	23	21
3	11	1	14	8	3	11	15	16	3	20	13	23	22
3	24	1	14	8	4	11	16	16	4	20	15	23	23
3	8			8	5	11	17	16	5	20	16	24	3
3	9			8	8	11	18	16	8	20	17	24	4
3	10			8	9	11	21	16	9	20	18	24	5
3	12			8	10	11	22	16	10	20	21		
3	13			8	12	11	23	16	11	20	22		
3	15			8	13	11	24	16	12	20	23		
3	16			8	15	13	2	16	13	20	24		

alue-4

4	9	1	1
9	22	1	1
7	9	1	2
17	22	1	2
10	23	1	3
15	23	1	3
25	23	1	3
14	22	1	4
24	22	1	4
2	11	1	5
5	10	1	5
5	12	1	5
10	25	1	5
15	25	1	5
19	11	1	5
5	1	1	0
25	5	1	2
3	25	1	6
3	1	1	6
4	25	1	7
4	1	1	7
5	25	1	8
5	1	1	8
6	25	1	9
6	1	1	9
7	25	1	10
7	1	1	10
19	22	1	11
1	4	1	11
20	23	1	12
14	4	1	12
20	25	1	13
1	11	1	13
22	22	1	14
8	2	1	14
24	11	1	15
19	4	1	15
1	5	1	15
3	20	1	15
25	10	1	16
12	1	1	16
25	25	1	17
16	1	1	17
6	17	1	17
16	6	1	18
3	14	1	18
1	3	1	19
14	18	1	19
18	2	1	20
8	14	1	20
22	1	1	21
18	15	1	21
6	12	1	21
2	22	1	22
12	22	1	22
2	6		
2	7		
2	8		
2	9		
2	10		
2	12		
2	13		
2	14		
2	15		
2	16		
2	17		
2	18		
2	19		
2	20		
2	21		
2	23		
2	24		
4	6		
4	7		
4	8		
4	10		
4	11		
4	13		
4	14		
4	15		
4	16		
6	6		
6	7		
6	8		
6	9		
6	10		
6	11		
6	13		
6	14		
6	15		
6	16		
7	6		
7	7		
7	8		
7	10		
7	11		
7	13		
7	14		
7	15		
7	16		
9	2		
9	3		
9	19		
9	20		
9	21		
9	23		
9	24		
10	2		
10	3		
11	2		
11	3		
11	19		
11	20		
11	21		
11	22		
11	23		
11	24		
12	19		
12	20		
12	21		
12	23		
12	24		
13	2		
13	3		
14	2		
14	3		
14	19		
14	20		
14	21		
14	23		
14	24		
15	2		
15	3		
16	19		
16	20		
16	21		
16	22		
16	23		
16	24		
17	2		
17	3		
17	19		
17	20		
17	21		
17	23		
17	24		
19	2		
19	3		
19	6		
19	7		
19	8		
19	9		
19	12		
19	13		
19	14		
19	15		
19	16		
19	19		
19	20		
19	21		
19	23		
19	24		
20	6		
20	7		
20	8		
20	9		
20	11		
20	12		
20	13		
20	14		
20	15		
20	16		
21	2		
21	3		
21	6		
21	7		
21	8		
21	9		
21	11		
21	12		
21	13		
21	14		
21	15		
21	16		
21	19		
21	20		
21	21		
21	22		
21	23		
21	24		
22	2		
22	3		
22	6		
22	7		
22	8		
22	9		
22	11		
22	12		
22	13		
22	14		
22	15		
22	16		
22	19		
22	20		
22	21		
22	23		
22	24		
23	2		
23	3		
23	6		
23	7		
23	8		
23	9		
23	11		
23	12		
23	13		
23	14		
23	15		
23	16		
24	2		
24	3		
24	6		
24	7		
24	8		
24	9		
24	12		
24	13		
24	14		
24	15		
24	16		
24	19		
24	20		
24	21		
24	23		
24	24		

terminalintensive-2

23	4	1	1
14	16	2	1
1	16	2	1
1	6	1	1
1	2	1	1
18	1	2	2
21	1	2	2
18	16	2	2
1	7	1	2
3	1	2	3
19	1	2	3
23	13	1	3
1	9	1	3
23	1	2	4
23	2	1	4
23	5	1	4
20	16	2	4
1	15	1	4
4	1	2	5
4	16	2	5
1	13	1	5
2	1	2	6
9	1	2	6
7	16	2	6
7	1	2	7
13	16	2	7
6	16	2	7
6	1	2	8
5	16	2	8
10	1	2	9
10	16	2	9
9	16	2	9
1	16	1	9
8	1	2	10
11	16	2	10
8	16	2	10
1	8	1	10
23	16	1	11
19	16	2	11
16	16	2	11
1	14	1	11
23	9	1	12
23	16	2	12
1	5	1	12
5	1	2	13
12	16	2	13
1	11	1	13
11	1	2	14
1	4	1	14
13	1	2	15
1	3	1	15
1	1	2	16
12	1	2	16
15	1	2	16
23	1	1	16
15	16	2	16
3	16	2	17
1	12	1	17
2	16	2	18
1	10	1	18
22	1	2	19
23	7	1	19
23	11	1	19
23	15	1	19
22	16	2	19
14	1	2	20
20	1	2	20
23	3	1	20
17	1	2	21
23	6	1	21
23	8	1	21
16	1	2	22
23	10	1	22
23	14	1	23
21	16	2	23
23	12	1	24
17	16	2	24

difficult-2

23	4	1	1	1	9	1	3	5	15	2	8	5	1	2	13	15	15	2	16	20	1	2	20
14	15	2	1	23	2	1	4	10	1	2	9	12	15	2	13	13	15	2	16	23	3	1	20
1	15	2	1	1	15	1	4	9	15	2	9	1	11	1	13	3	15	2	17	17	1	2	21
1	6	1	1	4	1	2	5	8	1	2	10	11	1	2	14	1	12	1	17	23	6	1	21
18	1	2	2	4	15	2	5	11	15	2	10	1	4	1	14	2	15	2	18	23	8	1	21
21	1	2	2	1	13	1	5	8	15	2	10	13	1	2	15	1	10	1	18	16	1	2	22
18	15	2	2	9	1	2	6	1	8	1	10	1	3	1	15	22	1	2	19	23	10	1	22
1	7	1	2	7	15	2	6	16	15	2	11	1	1	2	16	23	7	1	19	23	14	1	23
3	1	2	3	7	1	2	7	1	14	1	11	12	1	2	16	23	11	1	19	21	15	2	23
19	1	2	3	6	15	2	7	23	9	1	12	14	1	2	16	23	15	1	19	23	12	1	24
23	13	1	3	6	1	2	8	1	5	1	12	23	1	1	16	22	15	2	19	17	15	2	24

modifieddense-3

7	1	2	1	1	10	1	3	9	1	2	7	16	9	1	10	14	17	2	13	16	1	1	16
16	13	1	1	15	1	2	4	16	14	1	7	12	17	2	10	1	12	1	13	3	17	2	17
7	17	2	1	16	16	1	4	6	17	2	7	1	8	1	10	12	1	2	14	1	13	1	17
1	9	1	1	1	16	1	4	6	1	2	8	16	15	1	11	16	4	1	14	2	17	2	18
14	1	2	2	4	1	2	5	16	10	1	8	13	17	2	11	1	4	1	14	1	11	1	18
16	7	1	2	4	17	2	5	5	17	2	8	1	15	1	11	16	5	1	15	16	1	2	19
1	7	1	2	1	14	1	5	11	1	2	9	16	8	1	12	16	17	2	15	16	2	1	19
3	1	2	3	2	1	2	6	16	17	1	9	1	5	1	12	1	3	1	15	1	17	2	19
16	11	1	3	16	6	1	6	1	17	1	9	5	1	2	13	1	1	2	16	1	2	1	19
8	17	2	3	9	17	2	6	8	1	2	10	16	12	1	13	13	1	2	16				

moredifficult-2

22	4	1	1	1	9	1	3	5	15	2	8	5	1	2	13	15	15	2	16	22	3	1	20
14	15	2	1	22	2	1	4	10	1	2	9	12	15	2	13	13	15	2	16	17	1	2	21
1	15	2	1	1	15	1	4	9	15	2	9	1	11	1	13	3	15	2	17	22	6	1	21
1	6	1	1	4	1	2	5	8	1	2	10	11	1	2	14	1	12	1	17	22	8	1	21
18	1	2	2	4	15	2	5	11	15	2	10	1	4	1	14	2	15	2	18	16	1	2	22
21	1	2	2	1	13	1	5	8	15	2	10	13	1	2	15	1	10	1	18	22	10	1	22
18	15	2	2	9	1	2	6	1	8	1	10	1	3	1	15	22	1	2	19	22	14	1	23
1	7	1	2	7	15	2	6	16	15	2	11	1	1	2	16	22	7	1	19	21	15	2	23
3	1	2	3	7	1	2	7	1	14	1	11	12	1	2	16	22	11	1	19	22	12	1	24
19	1	2	3	6	15	2	7	22	9	1	12	14	1	2	16	22	15	2	19	17	15	2	24
22	13	1	3	6	1	2	8	1	5	1	12	22	1	1	16	20	1	2	20				

pedabox-2

1	16	2	1	1	1	2	4	1	4	1	8	7	16	2	12	15	16	2	16	13	16	2	20
1	14	1	1	14	16	2	4	15	5	1	8	6	1	2	13	4	1	2	17	15	12	1	21
1	15	1	2	3	1	2	5	4	16	2	9	1	9	1	13	1	2	1	17	10	16	2	21
1	12	1	2	1	3	1	5	1	16	1	9	15	4	1	13	5	1	2	18	8	16	2	21
15	15	1	2	15	3	1	5	8	1	2	10	15	6	1	13	1	1	1	18	6	16	2	22
9	1	2	2	15	7	1	6	1	8	1	10	15	8	1	14	5	16	2	19	15	11	1	22
7	1	2	3	1	5	1	6	1	6	1	10	13	1	2	14	2	16	2	19				
15	13	1	3	10	1	2	7	2	1	2	11	12	1	2	15	15	16	1	19				
3	16	2	3	1	10	1	7	1	11	1	11	15	9	1	15	9	16	2	19				
15	2	1	3	12	16	2	7	11	1	2	12	15	1	2	16	15	14	1	20				

augmenteddense-2

7	1	2	1	1	10	1	3	9	1	2	7	16	9	1	10	13	18	2	13	16	1	1	16
16	13	1	1	14	1	2	4	16	14	1	7	11	18	2	10	1	12	1	13	3	18	2	17
7	18	2	1	16	16	1	4	6	18	2	7	1	8	1	10	11	1	2	14	1	13	1	17
1	9	1	1	1	16	1	4	6	1	2	8	16	15	1	11	16	4	1	14	2	18	2	18
13	1	2	2	4	1	2	5	16	10	1	8	12	18	2	11	1	4	1	14	1	11	1	18
16	7	1	2	4	18	2	5	5	18	2	8	1	15	1	11	16	5	1	15	15	1	2	19
1	7	1	2	1	14	1	5	10	1	2	9	16	8	1	12	15	18	2	15	16	2	1	19
3	1	2	3	2	1	2	6	16	17	1	9	1	5	1	12	1	3	1	15	1	18	2	19
16	11	1	3	16	6	1	6	1	17	1	9	5	1	2	13	1	1	2	16	1	2	1	19
8	18	2	3	9	18	2	6	8	1	2	10	16	12	1	13	12	1	2	16				

gr2-8-32

9	2	1	1	9	1	1	2	9	6	1	4	2	9	1	6	7	9	1	8	6	4	1	
9	4	1	1	1	5	1	2	9	9	1	4	1	7	1	6	7	1	1	8	6	5	1	
5	9	1	1	9	5	1	3	7	6	1	4	2	1	1	7	3	9	1	8	6	6	1	
1	3	1	1	4	9	1	3	1	1	1	5	8	1	1	7	9	8	1	8				
1	8	1	1	8	9	1	3	5	1	1	5	1	2	1	7	5	4	1					
3	1	1	2	1	9	1	3	1	4	1	5	4	6	1	7	5	5	1					
4	1	1	2	9	3	1	4	9	7	1	6	6	9	1	8	5	6	1					

sb40-56

1	1	1	55	1	27	1	38	8	1	1	42	21	41	1	44	34	41	1	4	41	12	1	33
1	2	1	15	1	28	1	51	8	41	1	7	22	1	1	37	35	1	1	30	41	13	1	25
1	3	1	48	1	29	1	56	9	1	1	27	22	41	1	55	35	41	1	41	41	15	1	24
1	5	1	29	1	32	1	18	9	41	1	39	23	41	1	37	36	1	1	28	41	16	1	26
1	6	1	39	1	33	1	8	10	1	1	30	24	1	1	35	36	41	1	45	41	21	1	43
1	7	1	22	1	34	1	24	10	41	1	56	24	41	1	33	37	1	1	13	41	22	1	34
1	8	1	16	1	35	1	18	11	1	1	23	25	1	1	53	37	41	1	17	41	23	1	17
1	9	1	42	1	36	1	27	11	41	1	3	25	41	1	26	38	1	1	19	41	26	1	41
1	10	1	13	1	38	1	40	12	1	1	54	26	1	1	35	38	41	1	28	41	28	1	6
1	11	1	19	1	40	1	12	12	41	1	14	26	41	1	23	40	1	1	44	41	30	1	46
1	13	1	15	2	1	1	31	13	1	1	7	27	1	1	47	40	41	1	10	41	34	1	34
1	16	1	4	3	1	1	6	13	41	1	10	28	1	1	5	41	1	1	49	41	35	1	9
1	19	1	43	4	1	1	40	16	41	1	45	28	41	1	32	41	2	1	3	41	36	1	52
1	21	1	12	4	41	1	5	17	41	1	8	29	1	1	52	41	3	1	50	41	37	1	20
1	22	1	32	5	1	1	49	18	1	1	11	29	41	1	16	41	6	1	9	41	39	1	25
1	23	1	29	5	41	1	2	19	1	1	14	30	1	1	11	41	7	1	51	41	40	1	47
1	24	1	21	6	1	1	46	19	41	1	20	31	41	1	22	41	8	1	31	41	41	1	21
1	25	1	36	7	1	1	1	20	1	1	53	33	1	1	38	41	9	1	50				
1	26	1	54	7	41	1	48	20	41	1	36	33	41	1	1	41	11	1	2				

List of Figures

List of Tables

Bibliography

Programming and Software

A. V. Aho, R. Sethi, and J. D. Ullman. *Compiler Design: Principles, Techniques, and Tools*. Addison-Wesley, .

A. V. Aho and J. D. Ullman. *Foundations of Computer Science*. Freeman and Company, .

K. Beck. *Extreme Programming*. Addison-Wesley, .

K. Beck. *Test-Driven Development*. Addison-Wesley, .

J. Bentley. *More Programming Perls*. Addison-Wesley, .

J. Bentley. *Programming Perls*. Addison-Wesley, .

F. P. Brooks. *The Mythical Man-Month*. Addison-Wesley, anniversary edition, .

I. F. Darwin. *Checking C Programs with lint*. O'Reilly&Associates, .

S. C. Dewhurst. *C++ Gotchas*. Addison-Wesley, .

D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, ( ): – , .

A. I. Holub. *Compiler Design in C*. Prentice Hall, .

S. C. Johnson. Yacc – yet another compiler-compiler. Technical Report Computer Science # , Bell Laboratories, Murray Hill, New Jersey, .

S. H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley, nd edition, .

B. W. Kernighan and R. Pike. *The UNIX Programming Environment*. Prentice Hall, .

B. W. Kernighan and R. Pike. *The Practise of Programming*. Addison-Wesley, .

D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume . Addison-Wesley, nd edition, a.

D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume . Addison-Wesley, nd edition, b.

M. E. Lesk. Lex – a lexical analyzer generator. Technical Report Computer Science # , Bell Laboratories, Murray Hill, New Jersey, .

J. R. Levine, T. Mason, and D. Brown. *lex & yacc*. O'Reilly&Associates, nd edition, .

D. Libes. *Obfuscated C and other Mysteries*. Wiley, .

S. A. Maguire. *Writing Solid Code*. Mircosoft Press, .

B. Meyer. *Eiffel: The Language*. Prentice Hall, .

S. Meyers. *More Effective C++*. Addison-Wesley, .

S. Meyers. *Effective C++*. Addison-Wesley, nd edition, .

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, nd edition, .

A. T. Schreiner and G. Friedman. *Compiler bauen mit UNIX*. Hanser, .

R. Sedgewick. *Algorithms*. Addison-Wesley, nd edition, .

M. Shepperd. *Foundations of Software Measurement*. Prentice Hall, .

H. Sutter. *Exceptional C++*. Addison-Wesley, .

H. Sutter. *More Exceptional C++*. Addison-Wesley, .

A. S. Tanenbaum. *Modern Operating Systems*. Pearson Education, .

P. van der Linden. *Expert C Programming*. Prentice Hall, .

A. H. Watson and T. J. McCabe. Structured testing: A testing methodology using the cyclomatic complexity metric. Technical Report - , Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, USA, .

H. Zuse. History of software measurement. http://irb.cs.tu-berlin.de/˜zuse/metrics/3-hist.html, .

Mathematical Programming

E. D. Andersen and K. D. Andersen. Presolve in linear programming. *Mathematical Programming*, : – , .

E. Balas and C. Martin. Pivot and complement – a heuristic for / programming. *Management Science*, : – , .

E. Balas, S. Schmietab, and C. Wallacea. Pivot and shift – a mixed integer programming heuristic. *Discrete Optimization*, : – , .

J. Bisschop and A. Meeraus. On the development of a general algebraic modeling system in a strategic planning environment. *Mathematical Programming Study*, : – , .

R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, ( ): – , .

R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. MIP: Theory and practice – closing the gap. In M. J. D. Powell and S. Scholtes, editors, *System Modelling and Optimization: Methods, Theory and Applications*. Kluwer, .

R. E. Bixby and D. K. Wagner. A note on detecting simple redundancies in linear systems. *Operation Research Letters*, ( ): – , .

A. L. Brearley, G. Mitra, and H. P. Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical Programming*, : – , .

M. R. Bussieck and A. Meeraus. General algebraic modeling system (GAMS). In J. Kallrath, editor, *Modeling Languages in Mathematical Optimization*, pages – . Kluwer, .

V. Chvátal. *Linear Programming*. H.W. Freeman, New York, .

K. Cunningham and L. Schrage. The LINGO algebraic modeling language. In J. Kallrath, editor, *Modeling Languages in Mathematical Optimization*, pages – . Kluwer, .

G. B. Dantzig. The diet problem. *Interfaces*, : – , .

S. Elhedhli and J.-L. Go n. The integration of an interior-point cutting plane method within a branch-and-price algorithm. *Mathematical Programming*, : – , .

R. Fourer and D. M. Gay. Experience with a primal presolve algorithm. In W. W. Hager, D. Hearn, and P. Pardalos, editors, *Large Scale Optimization: State of the Art*, pages – . Kluwer, .

R. Fourer and D. M. Gay. Numerical issues and influences in the design of algebraic modeling languages for optimization. In D. F. Gri ths and G. Watson, editors, *Proceedings of the 20th Biennial Conference on Numerical Analysis*, number Report NA/ in Numerical Analysis, pages – . University of Dundee, .

R. Fourer, D. M. Gay, and B. W. Kernighan. A modelling language for mathematical programming. *Management Science*, ( ): – , .

R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modelling Language for Mathematical Programming*. Brooks/Cole—Thomson Learning, nd edition, a.

R. Fourer, L. B. Lopes, and K. Martin. LPFML: A W C XML schema for linear programming. Technical report, Department of Industrial Engineering and Management Sciences, Northwestern University, b. Information available at http://gsbkip.uchicago.edu/fml.

M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Bulletin no. 13*, pages – , . Data available at http://www.netlib.org/netlib/lp.

J. Gondzio. Presolve analysis of linear programs prior to applying an interior point method. *INFORMS Journal on Computing*, ( ): – , .

J. Gondzio and A. Grothey. Reoptimization with the primal-dual interior point method. *SIAM Journal on Optimization*, ( ): – , .

T. Hürlimann. The LPL modeling language. In J. Kallrath, editor, *Modeling Languages in Mathematical Optimization*, pages – . Kluwer, .

J. Kallrath. Mathematical optimization and the role of modeling languages. In J. Kallrath, editor, *Modeling Languages in Mathematical Optimization*, pages – . Kluwer, a.

J. Kallrath, editor. *Modeling Languages in Mathematical Optimization*. Kluwer, b.

T. Koch. The final NETLIB-LP results. *Operations Research Letters*, : – , .

A. Martin. Integer programs with block structure. Habilitations-Schrift, Technische Universität Berlin, Fachbereich Mathematik, .

C. Mészàros and U. H. Suhl. Advanced preprocessing techniques for linear and quadratic programming. *OR Spectrum*, : – , .

F. Plastria. Formulating logical implications in combinatorial optimization. *European Journal of Operational Research*, : – , .

M. W. P. Savelsbergh. Preprocessing and probing for mixed integer programming problems. *ORSA Journal on Computing*, pages – , .

H. Schichl. Models and the history of modeling. In J. Kallrath, editor, *Modeling Languages in Mathematical Optimization*, pages – . Kluwer, .

K. Spielberg. The optimization systems MPSX and OSL. In J. Kallrath, editor, *Modeling Languages in Mathematical Optimization*, pages – . Kluwer, .

J. A. Tomlin and J. S. Welch. Formal optimization of some reduced linear programming problems. *Mathematical Programming*, : – , .

J. A. Tomlin and J. S. Welch. Finding duplicate rows in a linear programming model. *Operations Research Letters*, ( ): – , .

H. P. Williams and S. C. Brailsford. Computational logic and integer programming. In J. E. Beasley, editor, *Advances in Linear and Integer Programming*, pages – . Oxford University Press, .

R. Wunderling. Paralleler und objektorientierter Simplex. Technical Report TR - , Konrad-Zuse-Zentrum Berlin, .

Telecommunications

K. Aardal, M. Labbé, J. Leung, and M. Queyranne. On the two-level uncapacitated facility location problem. *INFORMS Journal on Computing*, ( ): – , .

E. Amaldi, A. Capone, and F. Malucelli. Planning UMTS base station location: Optimization models with power control and algorithms. *IEEE Transactions on Wireless Communication*, ( ): – , a.

E. Amaldi, A. Capone, F. Malucelli, and F. Signori. UMTS radio planning: Optimizing base station configuration. In *Proceedings of IEEE VTC Fall 2002*, volume , pages – . .

E. Amaldi, A. Capone, F. Malucelli, and F. Signori. Optimizing base station location and configuration in UMTS networks. In *Proceedings of INOC 2003*, pages – . b.

A. Balakrishnan, T. L. Magnanti, and R. T. Wong. A decomposition algorithm for local access telecommunication network expansion planning. *Operations Research*, ( ): – , .

C. A. Balanis. *Antenna Theory: Analysis and Design*. Wiley, .

N. D. Bambos, S. C. Chen, and G. J. Pottie. Radio link admission algorithms for wireless networks with power control and active link quality protection. *IEEE*, pages – , .

H. L. Bertoni. *Radio Progation for Modern Wireless Applications*. Prentice Hall, .

D. Bienstock and O. Günlück. Capacitated network design—polyhedral structure and computation. *INFORMS Journal on Computing*, ( ): – , .

A. Bley. A Lagrangian approach for integrated network design and routing in IP networks. In *Proceedings of International Network Optimization Conference (INOC 2003), Evry/Paris, France, 2003*, pages – . .

A. Bley and T. Koch. Optimierung des G-WiN. *DFN-Mitteilungen*, pages – , .

A. Bley, T. Koch, and R. Wessäly. Large-scale hierarchical networks: How to compute an optimal hierarchy? In H. Kaindl, editor, *Networks 2004: 11th International Telecommunications Network Strategy and Planning Symposium, June 13-16, 2004, Vienna, Austria - Proceedings*, pages – . VDE Verlag, .

E. Brockmeyer, H. Halstrom, and A. Jensen. *The life and works of A. K. Erlang*. Academy of Technical Sciences, Copenhagen, .

A. Eisenblätter, A. Fügenschuh, E. Fledderus, H.-F. Geerdes, B. Heideck, D. Junglas, T. Koch, T. Kürner, and A. Martin. Mathematical methods for automatic optimization of UMTS radio networks. Technical Report D . , IST- - MOMENTUM, a.

A. Eisenblätter, A. Fügenschuh, H.-F. Geerdes, D. Junglas, T. Koch, and A. Martin. Optimization methods for UMTS radio network planning. In *Operation Research Proceedings 2003*, pages – . Springer, b.

A. Eisenblätter, H.-F. Geerdes, D. Junglas, T. Koch, T. Kürner, and A. Martin. Final report on automatic planning and optimisation. Technical Report D . , IST- - MOMENTUM, c.

A. Eisenblätter, H.-F. Geerdes, T. Koch, A. Martin, and R. Wessäly. UMTS radio network evaluation and optimization beyond snapshots. Technical Report – , Konrad-Zuse-Zentrum Berlin, .

A. Eisenblätter, H.-F. Geerdes, T. Koch, and U. Türke. Describing UMTS radio networks using XML. Technical Report IST- - -MOMENTUM-XML-PUB, MOMENTUM, d.

A. Eisenblätter, H.-F. Geerdes, T. Koch, and U. Türke. MOMENTUM public planning scenarios and their XML format. Technical Report TD( ) , COST , Prague, Czech Republic, e.

A. Eisenblätter, T. Koch, A. Martin, T. Achterberg, A. Fügenschuh, A. Koster, O. Wegel, and R. Wessäly. Modelling feasible network configurations for UMTS. In G. Anandalingam and S. Raghavan, editors, *Telecommunications Network Design and Management*. Kluver, f.

I. Gamvros and S. R. B. Golden. An evolutionary approach to the multi-level capacitated minimum spanning tree problem. In G. Anandalingam and S. Raghavan, editors, *Telecommunications Network Design and Management*. Kluver, .

H.-F. Geerdes, E. Lamers, P. Lourenço, E. Meijerink, U. Türke, S. Verwijmeren, and T. Kürner. Evaluation of reference and public scenarios. Technical Report D . , IST- - MOMENTUM, .

N. Geng and W. Wiesbeck. *Planungsmethoden für die Mobilkommunikation*. Springer, .

F. Gil, A. R. Claro, J. M. Ferreira, C. Pardelinha, and L. M. Correia. A -D interpolation method for base-station antenna radiation patterns. *IEEE Antenna and Propagation Magazine*, ( ): – , .

L. Hall. Experience with a cutting plane algorithm for the capacitated spanning tree problem. *INFORMS Journal on Computing*, ( ): – , .

H. Holma and A. Toskala. *WCDMA for UMTS*. Wiley, .

K. Holmberg and D. Yuan. A Lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, ( ): – , .

S. Jakl, A. Gerdenitsch, W. Karner, and M. Toeltsch. Analytical algorithms for base station parameter settings in UMTS networks. Technical Report COST TD( ) , Institut für Nachrichtentechnik und Hochfrequenztechnik, Technische Universität Wien, .

T. Kürner. Propagation models for macro-cells. In *Digital Mobile Radio towards Future Generation Systems*, pages – . COST Telecom Secretariat, .

R. Mathar and M. Schmeink. Optimal base station positioning and channel assignment for G mobile networks by integer programming. *Annals of Operations Research*, : – , .

P. Mirchandani. The multi-tier tree problem. *INFORMS Journal on Computing*, ( ): – , .

M. J. Nawrocki and T. W. Wieckowski. Optimal site and antenna location for UMTS – output results of G network simulation software. *Journal of Telecommunications and Information Technology*, .

K. Park, K. Lee, S. Park, and H. Lee. Telecommunication node clustering with node compatibility and network survivability requirements. *Managment Science*, ( ): – , .

S. Qiao and L. Qiao. A robuts and e cient algorithm for evaluating Erlang B formula, . http://www.dcss.mcmaster.ca/˜qiao/publications/erlang.ps.Z.

B. Rakoczi, E. R. Fledderus, B. Heideck, P. Lourenço, and T. Kürner. Reference scenarios. Technical Report D . , IST- - MOMENTUM, .

S. R. Saunders. *Antennas and Propagation for Wireless Communication Systems*. Wiley, .

K. Sipilä, J. Laiho-Ste ens, A. Wacker, and M. Jäsberg. Modeling the impact of the fast power control on the WCDMA uplink. In *IEEE VTS Proceedings of Vehicular Technology Conference*, pages – . Huston, .

R. Wessäly. *DImensioning Survivable Capacitated NETworks*. Ph.D. thesis, Technische Universität Berlin, .

R. M. Whitaker and S. Hurley. Evolution of planning for wireless communication systems. In *Proceedings of HICSS'03*. IEEE, Big Island, Hawaii, .

Steiner Trees and VLSI Layout

C. Boit. Personal communication. .

M. L. Brady and D. J. Brown. VLSI routing: Four layers su ce. In F. P. Preparata, editor, *Advances in Computing Research: VLSI theory*, volume , pages – . Jai Press, London, .

M. Burstein and R. Pelavin. Hierachical wire routing. *IEEE Transactions on computer-aided design*, : – , .

S. Chopra. Comparison of formulations and a heuristic for packing Steiner trees in a graph. *Annals of Operations Research*, : – , .

J. P. Coohoon and P. L. Heck. BEAVER: A computational-geometry-based tool for switchbox routing. *IEEE Transactions on computer-aided design*, : – , .

M. Grötschel, M. Jünger, and G. Reinelt. Via Minimization with Pin Preassignments and Layer Preference. *ZAMM – Zeitschrift für Angewandte Mathematik und Mechanik*, ( ): – , .

M. Grötschel, A. Martin, and R. Weismantel. Packing Steiner trees: A cutting plane algorithm and computational results. *Mathematical Programming*, : – , a.

M. Grötschel, A. Martin, and R. Weismantel. Packing Steiner trees: Further facets. *European Journal of Combinatorics*, : – , b.

M. Grötschel, A. Martin, and R. Weismantel. Packing Steiner trees: Polyhedral investigations. *Mathematical Programming*, : – , c.

M. Grötschel, A. Martin, and R. Weismantel. The Steiner tree packing problem in VLSI design. *Mathematical Programming*, ( ): – , .

D. G. Jørgensen and M. Meyling. *Application of column generation techniques in VLSI design*. Master's thesis, Department of Computer Science, University of Copenhagen, .

M. Jünger, A. Martin, G. Reinelt, and R. Weismantel. Quadratic / optimization and a decomposition approach for the placement of electronic circuits. *Mathematical Programming*, : – , .

T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, : – , .

B. Korte, H.-J. Prömel, and A. Steger. Steiner trees in VLSI-layout. In B. Korte, L. Lovász, H.-J. Prömel, and A. Schrijver, editors, *Paths, Flows, and VLSI-Layout*. Springer, .

T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley, .

W. Lipski. On the structure of three-layer wireable layouts. In F. P. Preparata, editor, *Advances in Computing Research: VLSI theory*, volume , pages – . Jai Press, London, .

W. K. Luk. A greedy switch-box router. *Integration*, : – , .

A. Martin. *Packen von Steinerbäumen: Polyedrische Studien und Anwendungen*. Ph.D. thesis, Technische Universität Berlin, .

T. Polzin. *Algorithms for the Steiner Problem in Networks*. Ph.D. thesis, Universität des Saarlandes, .

R. T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, : – , .

Miscellaneous

D. Applegate, R. E. Bixby, V. Chvátal, and W. Cook. Concord – combinatorial optimization and networked combinatorial optimization research and development environment. . http://www.math.princeton.edu/tsp/concorde.html.

R. Borndörfer. *Aspects of Set Packing, Partitioning, and Covering*. Ph.D. thesis, Technische Universität Berlin, .

J. Borwein and D. Bailey. *Mathematics by Experiment*. A K Peters, .

S. Chopra and M. R. Rao. The partition problem. *Mathematical Programming*, ( ): – , .

V. Chvátal. A farmer's daughter in our midst: An interview with Vašek Chvátal. . http://www.cs.rutgers.edu/~mcgrew/Explorer/1.2/#Farmers.

G. Dueck. Mathematik, fünfzehnprozentig. *DMV-Mitteilungen*, pages – , .

C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. Formulations and valid inequalities for the node capacitated graph partitioning problem. *Mathematical Programming*, : – , .

C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. The node capacitated graph partitioning problem: A computational study. *Mathematical Programming*, : – , .

M. R. Garey and D. S. Johnson. *Computers and Intractability*. H.W. Freeman, New York, .

G. C. F. Greve. Brave GNU World. *Linux Magazin*, pages – , .

M. Grötschel and Y. Wakabayashi. Facets of the clique partitioning polytope. *Mathematical Programming*, ( ): – , .

D. S. Johnson. A theoretician's guide to the experimental analysis of algorithms. In M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, volume of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages – . American Mathematical Society, .

F. Ortega and L. A. Wolsey. A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. *Networks*, ( ): – , .

R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, : – , .

A. Schrijver. *Combinatorial Optimization*. Springer, .

R. Sosič and J. Gu. , , million queens in less than a minute. *SIGART Bulletin*, ( ): – , .

P. Wessel and W. H. F. Smith. The generic mapping tools (GMT), version . . . Code and documentation available at http://gmt.soest.hawaii.edu.