Decision support by combinatorial optimization : case studies

Decision support by combinatorial optimization

case studies

Decision support by combinatorial optimization

case studies

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr. M. Rem, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 21 december 1999 om 16.00 uur

door

Sergey Romualdovich Tiourine

geboren te St.-Petersburg, Rusland

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. J.K. Lenstra
en
prof.dr. J. Wessels

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Tiourine, Sergey R.

Decision support by combinatorial optimization : case studies / by Sergey R. Tiourine. - Eindhoven : Eindhoven University of Technology, 1999.
Proefschrift. - ISBN 90-386-0721-0
NUGI 811
Subject headings: operations research / methodology / operations research : case studies
1991 Mathematics Subject Classification : 90 B 50, 90 B 90

Printed by Universiteitsdrukkerij Technische Universiteit Eindhoven

Acknowledgments

My Ph.D. project at Eindhoven University of Technology was certainly a good school of life for me, but as with any school one is happy when it is finally over. I am very glad that I managed to survive the advising efforts of Cor Hurkens and Jan Karel Lenstra. However, I am grateful to Jan Karel Lenstra for initiating consultancy projects, which became my battlefields, and for his involvement at a later stage of preparation of this manuscript.

I want to express my gratitude to all my colleagues, managers and engineers with whom I worked at various projects. They fascinated me with practical decision support, they taught me to see the issues, which I have just touched in this manuscript and whose solutions are yet to be discovered.

I cannot possibly express here how much do I appreciate the assistance of my friends, for it is due to them that everything became possible. My special thanks go to my psychotherapist Ms. R. M. Schönthal for educating me in the human nature.

Contents

Chapter 1

Introduction

Efficiently running a complex organization such as a manufacturing plant or an airline requires the precise coordination of materials, equipment, and people. Operations research analysts help organizations coordinate and operate in the most efficient manner by applying mathematical principles to organizational problems. Managers then evaluate alternatives and choose the course of action that best meets their goals.

Operations research (OR) is a science of mathematical modeling and solution of industrial decision problems. It tackles a wide range of issues facing large business and government organizations, including strategy, forecasting, resource allocation, facilities layout, inventory control, personnel schedules, and distribution systems. The history of OR goes back to the 1930's and 1940's, when pioneers like Leonid Vitaliyevich Kantorovich and George B. Dantzig set important precedents of systematic application of mathematical tools and mentality to modeling of complex economic and military logistic problems.

The process of OR intervention generally includes constructing an analytical abstraction of the essence of a real-life situation, solving it with mathematical tools, often implementing this solution as a computer program, and interpreting the solution back in terms of the original situation. Implementation of this process usually involves the following concepts:

- establishing direct contact with the key players in the decision making process;
- verbal problem formulation;
- data collection;
- mathematical model formulation;
- model analysis and validation;
- development of mathematical solution techniques;
- analysis of solution techniques;
- design, implementation and testing of software;
- analysis of results and formulation of an advice;
- implementation of the advice within the existing decision making process;
- reporting;
- maintenance.

Realization of these concepts is unthinkable without close cooperation of an OR analyst with managers and engineers, with decision makers and technicians, with those who possess the essential knowledge about the problem being modeled and with those whose daily work may be affected by the solutions suggested by an OR analyst. Some witty ideas about what it takes to make an OR project a success story have been given by Murphy (1998).

The mathematical tools of OR are usually those of statistics, stochastic processes and optimization. My experience is primarily, and in this thesis exclusively, concerned with the application of techniques from combinatorial optimization, but I believe that some of the conceptual statements made in this thesis are valid for practical OR in general.

In this thesis I summarize and make an attempt to generalize my experience based on three full-scale consultancy projects and one research project. The emphasis lies on empirical rules of model building and validation, and on criteria of algorithm selection. The interest in this kind of work in the OR community is expressed for example by Midgley (1998) and ILOG (1997).

The following two sections present my view on the practical aspects of mathematical modeling and solution techniques of combinatorial optimization. Section 1.3 contains the abstracts of the case studies, which link their full presentation in Chapters 2–5 with the preceding sections. Conclusions are given in Chapter 6.

1.1 Mathematical modeling

> As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.
> –Einstein (1921)

The purpose of modeling in general is to build an adequate representation of the process under study. The models in OR are built using mathematical relationships, such as equations, inequalities, etc., which correspond to some more down-to-earth relationships in the real world, like physical laws, technological constraints, etc. Williams (1978) gives a number of motives for building such a model:

- The actual exercise of building a model often reveals relationships which were not apparent to many people. As a result a greater understanding is achieved of the object being modeled.
- Having built a model it is usually possible to analyze it mathematically to help suggest courses of action which might not otherwise be apparent.
- Experimentation is possible with a model whereas it is often not possible or desirable to experiment with the object being modeled.

It is important to emphasize that a model is bound to be an approximation of the real process being modeled and that different models can be created of the very same process. A problem then is how to select an appropriate model which retains the most significant features of the process under study, but is devoid of all its redundant details. Clearly, a

methodology of model building and validation falls beyond the domain of mathematics and has to be sought in the so-called empirical sciences. The fundamental difference between mathematics and empirical sciences, such as physics, chemistry, psychology and economics, is that the validity of the statements in the latter is verified by observations of the actual phenomena, whereas in mathematics this is obviously not the case. This section is about my very practical view of this issue.

A mathematical model is defined by the relationships which it incorporates. These relationships are, to a large extent, independent of the input data in the model. In this respect, a model has to be distinguished from its instances, i.e., particular realizations of the constants and parameters of the model. A model can be used on many different occasions, whereas an instance is a representation of a particular process.

Many standard models are used in OR. A prominent place among them belongs to the models of combinatorial optimization, my area of specialization. I will start with a short description of combinatorial optimization models and their applications. A typical instance of a combinatorial optimization problem is formulated as follows: given a set of deterministically defined decisions, find the best among them with respect to certain criteria. Or formally:

Definition 1.1. Given a *set of feasible solutions* $\mathcal{I}$ and a *cost function* $f : \mathcal{I} \to \mathbb{R}$, find an element I^* with

$$f(I^*) = \min\{f(I) | I \in \mathcal{I}\}.$$

□

Such a general optimization problem is of little use unless there is a reasonable characterization of the set of feasible solutions $\mathcal{I}$ and an algorithm to evaluate the objective function for each $I \in \mathcal{I}$. The solution set is often represented by a set of decision variables, whose values are taken from certain domains. An assignment of values to the decision variables corresponds to a solution. The instance $(\mathcal{I}, f)$ is generally not given explicitly, i.e., by listing all solutions and their costs. Usually, an implicit description in terms of decision variables, constraints and a cost function defined on these variables, is provided, which allows to verify whether a certain solution belongs to $\mathcal{I}$ and to evaluate its cost in polynomial time.

A real-world decision problem can be modeled as a classical combinatorial optimization problem if it satisfies the following conditions:

- The problem allows for a good deterministic approximation, i.e., elementary decisions, their consequences and restrictions are known in advance and are to a certain degree deterministic.
- At least some of the elementary decisions are taken from discrete sets.
- There are clear, quantifiable objectives of the decision making process.

These assumptions are in fact quite restrictive. They suggest a very rigid concept of reality, which has room only for a purely deterministic cause-effect relationship (Ackoff, 1979). A leap forward in this respect is realized by stochastic integer programming (Kall and Wallace, 1994; Stougie and Van der Vlerk, 1997) and online optimization (Ascheuer et al., 1998)

models, which provide a way to cope with such factors as uncertainty and incompleteness of data. On the other hand, it does not mean that the classical combinatorial optimization approach cannot be successfully applied to a variety of real-life problems.

In this thesis, I use combinatorial optimization models for operational problems from production sequencing, telecommunication, statistical disclosure control and industrial cutting. Creating useful decision support tools was my primary objective in each of these exercises. Therefore, I was especially concerned with matters of model building and validation.

Developing a mathematical model is a long and interactive process. During this process different models are tried, verified, refined, again verified and possibly rejected. It is important to develop a model systematically, following a certain strategy. For the moment, I assume that a model is being built with an open mind, and that such factors as the existence of software packages or a project budget play no role. My personal favor then goes to the strategy suggested by Hax and Candea (1984). They advocate to move from simple to complex models. There are a lot of arguments in favor of and against this strategy. First of all, proposing a simple model can make one look unsophisticated in the eyes of the client, but it can be remedied by providing sufficient explanation about the approach being taken. Another argument against this approach is that intuitively one is trying to build a presumably complex model which best fits the first description of the problem. Unfortunately, the first description of a problem more often than not has little to do with the actual problem, and a consultant has to find a way to go beyond the symptoms to the essence of the client's problem. It is, therefore, very useful to perform at least one iteration to bring the positions of the two parties closer to each other. In this sense, the use of a simple model is more than justified, not only because of its relatively easy analysis, but also because of its superior illustrative power. Comparing various simple models with each other can often bring insight into the most important factors and relationships in the problem.

But before one even starts with mathematical modeling, it helps to get a broader view of the problem, for example by asking oneself the following simple questions:

- What are the goals of mathematical modeling?
- Which means are available to achieve these goals?
- Which precision is required in achieving the goals?

In production scheduling, a goal may be to optimize performance of a factory for the sake of effectiveness of production. The means may include changing the production sequence of jobs, modifying their release dates, and introducing extra working shifts. Precision can vary from seconds, in case of a fully automatic production line, to days and months, in case production involves research and development activities.

Moreover, these questions can easily lead to a conclusion that the problem is too vague to be approached by means of mathematical modeling, and an extra study is needed to quantify the objectives or to specify the means.

Having a certain strategy in mind, one may consider various tactics for its execution. It does not seem unreasonable, having tried a very simple model first, to try some very

general model of the same type to ensure that reality is "caught" between these two models. I must admit that it never occurred to me before to try something like this. Instead, I usually try to analyze which important factors are missing in the simple model, once it looks promising to serve as a basis.

As an illustration of the last point, consider the case described in Chapter 2. In this study of a production process I departed from the standard job shop scheduling model. After some tests and discussions I arrived at the conclusion that it can serve as a solid basis for my model, representing the most important relationships between decision variables in the problem. As the next step, I singled out a number of factors that seemed missing in the model, and if included in the model were likely to improve its fitting to the available data about the production process. Among these factors there were sequence-dependent set-up times, release and due dates. Later, I included these factors into the model and validated it again. The fitting was better, which unexpectedly resulted in a stream of new ideas and factors produced by the managers, which essentially made me start all over again. Besides the headache, it has also given me the idea of writing this thesis, because this stunning development found me absolutely unprepared and the standard textbooks left me wondering about what to do next. In physics, for example, it does not occur to many that one day the Manager can change His mind about the reality in His factory.

It is very difficult, if not impossible, to describe model construction as a sequential procedure. Instead, I will simply list the arguments, which I believe important to keep in mind during model development.

- The model has to reflect the goals, the means and the precision specified. It does not necessarily have to achieve them, because an evidence of unachievable goals, insufficient means or excessive precision set for the model may be a valuable result of mathematical modeling.
- The model has to be solvable in principle if it is to be analyzed algorithmically. Availability of solution techniques, their efficiency and compliance with the goals, means and precision set for the modeling, are important factors in this case.
- The model should not exaggerate its required precision.

This may sound trivial. Still, when one sets out to model reality, it is a good idea to stick to these principles. In practice, however, this path is often abandoned, for example, for the sake of the beauty of mathematical analysis, and comfort of the ideal world of mathematics as opposed to the uncertain and obscure reality. Some even dare to suggest that OR nowadays is only interesting as a branch of pure mathematics (Ackoff, 1979).

A funny example of the third point above I saw on the Melkunie milk packs in the Netherlands, where the expiration date was specified up to seconds. If the objective of this information was to reassure the customer, then it obviously failed to reach its purpose in my case.

Once the model has been constructed, it has to be validated. This is probably the most important step in model development. It is a pity when all this mobilization of intuition, analytic and synthetic thinking goes in vain, because one does not check whether the result is an adequate model of reality. Suppose one would be invited to fly on a gorgeous new

airplane, with shining wings and a proud tail, which never actually made it yet into the skies. I would not be among the volunteers. What is the value of a decision suggested by a reasonably looking model which has not been thoroughly validated? My strong impression is that the importance of model validation has been underestimated in OR. It is based on the study of practical OR literature, conversations with colleagues, and my own experience as a consultant. I am certainly not the first to realize the problem; see for example the paper of Barnett (1994). However, the following remark on the methodology of mathematical modeling by Pollock and Maltz (1994),

> "*If possible*, verifying to see if the model is accurate by testing it with actual data" [italics are mine, ST],

suggests that model validation is an option rather than a necessity, and may be interpreted as a permission to go ahead with an untested model.

Model validity is a multifaceted concept. First, it is necessary to validate whether the goals, means and precision required for the model have been properly met. Second, correctness of the analysis itself has to be validated. At last, the question to what extent the results of the modeling can be generalized, in the practical sense of extrapolated, has to be answered. Correctness of the algorithmic analysis will be briefly discussed in the next section. There is also the related question of software testing and verification. This question is a hot topic in software engineering (Yeh et al., 1992; Clarke and Wing, 1996), but it falls beyond the scope of this thesis.

Application of model validation techniques does not protect the model from failure in practice. It rather gives a sound evidence that the model can be trusted, at least under the circumstances of the tests. I will just provide a brief summary of techniques which I tried myself and found useful in practice. Examples of applications of these techniques can be found throughout Chapters 2-5.

- *Face validity test* (Hermann, 1967; Emshoff and Sisson, 1970) gives the initial impression of a model's realism, which is obtained by asking people who know the real system (managers, engineers, etc.) to judge whether the model and its results are reasonable.
- *Test of structural validity* (Barnett, 1994) studies the way a model reflects the real system performance. It starts with identifying all stated and implied assumptions, listing all decision variables and hypothesized relations between them. It proceeds with testing the assumptions and hypotheses against the data available about the real system. Importance of this test should not be underestimated, despite Milton Friedman's assertion (Constantinides and Malliaris, 1995) that a model should not be judged by the relevance of its assumptions, but rather by the realism of its predictions.
- *Computer simulations* (Law and Kelton, 1991; Kleijnen and Van Groenendaal, 1992) and *experiments* (Montgomery and Runder, 1999) are used to test the predictive power of a model. Simulations are used to test the model predictions against the data already acquired from the real system, while experiments are used when such data is scarce. *Field tests* in the military (Kleijnen and Alink, 1992) are typical examples of experiments in OR.

- *Sensitivity analysis* (Dantzig and Thapa, 1997) tests dependency of conclusions of a model on variations of assumptions and uncertainties in the data.

For other interesting reading about OR methodology I refer to Keys (1991), Flood and Carson (1990), Daellenbach et al. (1983) and Gass (1983).

It is important to realize that while performing an OR project it is virtually impossible to build a mathematical model absolutely objectively. A consultant of a company that sells linear programming software will be tempted to see the world through the spectacles of linear programming, someone fascinated by interactive systems will have a different view than me, who at times believes that everything can be combinatorially optimized. I will mention some factors that are contributing to the bias in mathematical modeling, and I hope that just by being aware of them one can become more objective:

- mathematical specialization;
- ease or beauty of algorithmic analysis;
- temptation to stick to secure techniques;
- availability of ready-to-use computer software;
- project budget.

Corbett et al. (1995) attempt to classify the bias in mathematical modeling in terms of so-called *strands of practice*, i.e., specialization in practicing certain techniques. They provide four examples of how different practitioners develop a sort of specialization in order to sustain the pressure of the market. They argue that the problem-oriented approach to a consultancy project, such that a customized solution is devised to the problem at hand by using whatever techniques are best suited, does not correspond to the reality of the market. Their opinion is that a consultant has to develop his or her strand of practice in order to be able to respond adequately to the demands of his clients. I find this to be a gloomy development. While application of prefabricated models and techniques seems appropriate on the operational level of decisions, on strategic level only a truly problem-oriented approach can help to alter the often cited opinion of managers that OR provides elegant solutions to nonexistent problems; see for example Ackoff (1979).

I will close this section by quoting the conclusions of Tilanus (1985), who summarizes the experience of 36 case studies in the Netherlands:

- there is still a lot of OR/MS work to be done, building models that fit problems better;
- quick and clean work, cutting out simple and flexible models, leads to success;
- a soft, friendly approach, involving and informing the user, is crucial.

1.2 Solution techniques

This section is a short practical introduction into the solution techniques of combinatorial optimization. It summarizes my experience with the practical design and implementation

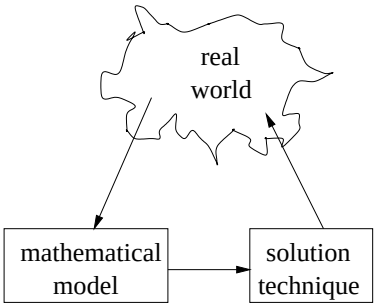


Figure 1.1: Solution cycle of a real-world problem.

of algorithms and reflects on the comparative study of algorithmic techniques within the CALMA project, described in Chapter 3.

Solution techniques are applied to mathematical models in order to generate mathematical solutions, which later can be interpreted in terms of courses of actions in a real-life situation being modeled. The solution cycle of a real-world problem by mathematical means is roughly sketched in Figure 1.1. The aim of an OR analyst is to perform this cycle with a minimum loss of precision and to suggest courses of actions which are close to the best possible ones in a given real-world situation. A loss of precision may occur at each stage. Firstly, a mathematical model is by its very nature not an exact representation of the corresponding real-world process. The mistake occurring at this stage is usually difficult to estimate in OR; here some of the model validation techniques described in the previous section may be useful. Secondly, it is not always possible to solve a mathematical model to optimality. Sometimes, the optimality gap of a generated solution, i.e., the difference between its value and the optimal value, can be estimated, but often it is simply taken for granted that a solution generated is close to optimal. Several techniques to estimate the solution quality are discussed in this section. At last, interpreting a mathematical solution in terms of the real-world problem may also result in a loss of precision.

Mathematical solution techniques are applied to mathematical models, not to real-life problems. Nonetheless, I will use the term *practical mathematical model* to describe a mathematical model together with all relevant practical limitations of an OR project. Such limitations can play an essential role in the algorithm selection process. For example, a practical job shop scheduling model may include restrictions on a solution technique such as a hard deadline on running or development time, or the requirement of high built-in flexibility. Therefore, the most efficient algorithm for the job shop scheduling problem may not be the choice for a practical variant of this problem.

Solution techniques in combinatorial optimization are algorithmic, i.e., solutions have to be obtained as the result of a computational procedure. Algorithms may vary from rules of thumb to extremely complex and elegant constructions with sophisticated quality guarantees. From a practical point of view, they are characterized by solution quality, running time, flexibility and range of applicability, development time and cost.

The algorithms in combinatorial optimization can be classified into the following groups:

- generating algorithms,
  - optimization algorithms,
  - approximation algorithms,
- evaluating algorithms.

Generating algorithms construct a solution with specified properties. For example, a tabu search algorithm will typically generate a good locally optimal solution. This group is comprised of *optimization* and *approximation* algorithms. Optimization algorithms find a guaranteed optimal solution for a given instance, while approximation algorithms try to find a good approximation of it. *Evaluating algorithms*, on the other hand, simply compute the cost for a given set of solutions. Simulation is a typical example of an evaluating algorithm.

One of the essential criteria of algorithm performance is the running time. As the actual running time of an algorithm may differ significantly from computer to computer, algorithmic complexity is expressed in terms of the number of elementary steps, like comparisons and arithmetic operations, required for the execution of the algorithm on a hypothetical computer. The *complexity* of an algorithm for an input of a given size n is defined as the worst-case behavior of the algorithm over all inputs of size n. Typically, algorithms are classified into two major groups according to their complexity. *Polynomial-time* algorithms are those algorithms for which the growth of complexity in n is bounded by a polynomial in n; other algorithms are called *superpolynomial-time*. A polynomial-time bound on the running time of an algorithm is often considered as an indication of its efficiency and practical applicability. Although the correlation between this theoretical measure of performance and the practical efficiency of an algorithm gains more and more evidence, the complexity of practical implementations of algorithms remains largely unknown. For example, efficient implementations of algorithms for linear programming are full of tricks which obscure the analysis, yet speed up the code (Todd, 1994; Lustig et al., 1994). For an introductory book on algorithms and complexity in combinatorial optimization see Papadimitriou and Steiglitz (1998).

The complexity theory of mathematical models (Garey and Johnson, 1979; Papadimitriou, 1994) tries to explain the observed phenomenon that for some models, like weighted matching and linear programming, there are efficient algorithms, while for others, like satisfiability and integer linear programming, no such algorithms are known. This theory knows a vast number of different complexity classes. The two most widely used ones are $\mathcal{P}$ and $\mathcal{NP}$, which can be loosely defined as the class of all yes/no problems solvable in polynomial time and the class of such problems solvable by polynomial-depth backtrack search, respectively. The most difficult problems in $\mathcal{NP}$ are called $\mathcal{NP}$*-complete* and their optimization variants are called $\mathcal{NP}$*-hard*. Complexity theory provides strong evidence against the existence of a polynomial-time optimization algorithm for any $\mathcal{NP}$*-hard* problem. An informal introduction into the complexity theory of combinatorial optimization problems is given by Lenstra and Rinnooy Kan (1979).

It has become good practice to check whether a given problem belongs to one of these complexity classes. In most, but certainly not all, practical applications it is a matter of routine to check that the problem at hand is $\mathcal{NP}$*-hard*. If this is indeed the case, then it can

at best serve as an argument for opting for an approximation algorithm. However, there are plenty of examples of the successful application of optimization algorithms to practical $\mathcal{NP}$-*hard* problems. On the other hand, if a problem is solvable in polynomial time, a proven polynomial-time optimization algorithm may not be the best choice to tackle it in practice.

As most of the practical problems are $\mathcal{NP}$-*hard*, it is highly unlikely that any efficient algorithm can guarantee optimality of solutions generated for these problems. Therefore, one has to choose between an algorithm that runs fast or one that guarantees optimality of solutions. If this dilemma is resolved in favor of the former option, then another question arises: how well will the chosen approximation algorithm perform on the kind of instances for which it is intended? There are several techniques known to answer this question. Traditionally, the algorithms are analyzed on basis of their *worst-case* behavior (Johnson and Papadimitriou, 1985). Here, the performance of an algorithm on the worst possible instance is studied, which provides an estimate valid for all instances. Another option is to assume some probability measure on the instances and to apply *average-case* analysis (Karp and Steele, 1985) to obtain an indication of the algorithm's performance on the most typical instance. *Empirical* analysis (Golden and Stewart, 1985; Johnson and McGeoch, 1997) is a very useful technique of algorithm analysis from a practical point of view. It studies the computational performance of algorithms on a representative set of instances. An example of an empirical study of algorithms is given in Chapter 3. However, neither empirical nor average-case analysis provide an estimate valid for all instances, and therefore one has to be careful in interpreting the results of these analyses for a particular instance. *Competitive* analysis (Ascheuer et al., 1998) studies the worst-case performance of an online algorithm and compares it with the corresponding offline optimal value.

Although each of the techniques for algorithm analysis has its advantages and drawbacks, these techniques should be considered as complementary to each other rather than competing. Empirical analysis is very effective if the set of instances is indeed representative of the real-life situations in which the algorithm is expected to function, but it can be misleading otherwise. Worst-case analysis often provides a too pessimistic picture of the algorithm's performance in practice, but it holds for any instance, including the most pathological ones. Average-case analysis of an algorithm gives a performance indication on the most typical instance. However, the results of this type of analysis depend on the probability distribution assumed on the set of instances. In practice, the choice of such a distribution is often dictated by the possibilities of an analysis.

The following are the most widely used solution techniques for practical combinatorial optimization models:

- optimization algorithms:
  - general enumerative techniques:
    - branch-and-bound,
    - constraint programming,
    - dynamic programming,
  - LP-based algorithms:

    - cutting plane,
    - column generation,
    - branch-and-cut,
    - branch-and-price,
- approximation algorithms:
  - with use of lower bounds:
    - LP-based algorithms,
    - Lagrangean relaxation based algorithms,
  - dispatching and other constructive rules,
  - local search:
    - iterative improvement,
    - tabu search,
    - simulated annealing,
    - variable-depth search,
    - genetic algorithms,
    - neural networks,
- simulation.

Branch-and-bound (Lawler and Wood, 1966; Balas and Toth, 1985) is the most generic optimization technique used in combinatorial optimization. This algorithm repeatedly partitions the solution space into smaller subsets and calculates lower bounds (assuming we have a minimization problem) for the objective value over each subset. The bounds are obtained by solving an easier problem over a given subset, so that the solution value of the new problem bounds the value of the original problem from below. The subsets for which the bound is at least equal to the cost of the best known feasible solution are eliminated from the rest of the search process. The remaining subsets are partitioned in turn until the cost of the best feasible solution will be no greater than the bound for any subset. The best solution found during this procedure is a global optimum.

The essential ingredients of any branch-and-bound algorithm for a combinatorial optimization problem P of the form $z_P = \min\{f(x)|x \in S\}$ include

- a lower bounding problem L: $z_L = \min\{g(y)|y \in T\}$, so that $z_L \leq z_P$;
- an algorithm to solve L;
- a branching rule, i.e., a procedure for partitioning the current subproblem P_i into subsets $S_{i1}, ..., S_{iq}$ such that $\bigcup_{j=1}^{q} S_{ij} = S_i$;
- a subproblem selection rule.

Additional features include

- an algorithm for calculating an upper bound for each subset;
- an algorithm for checking logical implications of the constraints and bounds to reduce the variable domains, or to tighten constraints.

General lower bounding techniques include relaxation and duality. A problem R of the form $z_R = \min\{h(x)|x \in W\}$ is called a *relaxation* of problem P if $S \subseteq W$ and $f(x) \geq h(x), \forall x \in S$. The fundamental property of a relaxation is that if R is infeasible, then so is P, and otherwise z_R is a lower bound on z_P.

Suppose that a problem P can be formulated as an *integer linear program*:

$$\begin{array}{lllll} \text{ILP:} & \min & cx & & \\ & \text{s.t.} & Ax & \leq & b, \\ & & x & \geq & 0, \quad \text{integer}, \end{array}$$

where A, b, and c have integer coefficients. Then the following formulation is called a *linear programming* relaxation of P:

$$\begin{array}{lllll} \text{LP:} & \min & cx & & \\ & \text{s.t.} & Ax & \leq & b, \\ & & x & \geq & 0. \end{array}$$

LP relaxation is the most popular way of obtaining lower bounds in combinatorial optimization. This is not only because there are theoretically efficient algorithms to solve LP, but also because there are powerful implementations which are able to solve LP's of practical size. However, it is often not enough to simply formulate an LP relaxation to be able to obtain a lower bounding procedure useful in a branch-and-bound algorithm. Sometimes, more work has to be done in order to strengthen such a relaxation. This is typically done by adding *valid* inequalities to LP, i.e., inequalities that are valid for any solution of ILP, but may be violated at an optimal solution to LP. An algorithm that implements this strategy is called a *cutting plane* algorithm and a corresponding branch-and-bound algorithm is called *branch-and-cut* (Nemhauser and Wolsey, 1988; Aardal and Van Hoesel, 1996). Algorithms of this type have been proven to be practically efficient for a wide variety of problems (Jünger et al., 1995; Aardal and Van Hoesel, 1999). Moreover, the design and implementation of branch-and-cut algorithms is not reserved to the experts in the field anymore, mainly due to the existence of specialized software packages supporting such algorithms like ABACUS (Thienel, 1995) and MINTO (Nemhauser et al., 1994).

Another widely used way of obtaining lower bounds makes use of *Lagrangean* relaxation of ILP (Nemhauser and Wolsey, 1988):

$$\begin{array}{lllll} \text{LR}(\lambda): & \min & cx + \lambda(A_1x - b_1) & & \\ & \text{s.t.} & A_2x & \leq & b_2, \\ & & x & \geq & 0, \quad \text{integer}, \end{array}$$

where $A = \binom{A_1}{A_2}$, $b = \binom{b_1}{b_2}$ and $\lambda \geq 0$. Typically, LR(λ) is chosen to be an easy problem with an efficient algorithm to solve it. For this purpose, a set of complicating constraints $A_1x \leq b_1$ is relaxed and the term $\lambda(A_1x - b_1)$ is introduced into the objective function to penalize the violation of these constraints. The lower bound given by LR(λ) is strengthened by maximizing LR(λ) over $\lambda \geq 0$, using for example multiplier adjustment (Held and Karp,

1970; Held and Karp, 1971) or subgradient optimization (Geoffrion, 1974; Held et al., 1974; Fisher, 1981).

Lower bounds can also be obtained using linear programming *duality* (Nemhauser and Wolsey, 1988). The following problem is called a *weak dual* of ILP:

$$\begin{array}{lrcl} \text{DP:} & \max & ub & \\ & \text{s.t.} & uA & \leq & c, \\ & & u & \leq & 0. \end{array} \tag{1.1}$$

The fundamental result of linear programming duality theory is that any feasible solution to DP gives a lower bound on the optimum of ILP, and if DP has an unbounded objective value, then ILP is infeasible.

Sometimes, ad hoc lower bounds can be derived for the problem at hand, which are neither based on a relaxation, nor on a dual of the original problem. For example, a certain objective value, say $\hat{z}_P$, is conjectured to correspond to a feasible solution to P. Subsequently, one tries to use logical implications to disprove the existence of a feasible solution with this objective value or lower, and thus proving that $\hat{z}_P+1$ is a lower bound on the optimum value z_P. An example of an application of this idea to a frequency assignment problem is due to Kolen and Van Hoesel (1995); it is dealt with in Chapter 3. Another example of a similar lower bounding technique for the job shop scheduling problem is given by Martin and Shmoys (1996).

A different approach is presented in Chapter 3. There, an ad hoc lower bound is derived for a frequency assignment problem using a non-linear lower bounding problem, which is solved by strong preprocessing and enumeration.

For other elements of a branch-and-bound algorithm, I refer to Linderoth and Savelsbergh (1999) for an excellent computational study of various branching and subproblem selection rules. An upper bound in a branch-and-bound algorithm can be calculated using any of the approximation algorithms discussed below. The issue of using logical implications to tighten constraints and to reduce the variable domains is addressed by Savelsbergh (1994) and Tsang (1993).

Developing a branch-and-bound algorithm, and in fact any optimization algorithm, requires a lot of insight in both the method and the problem. The result of such an exercise is usually unpredictable, in the sense that it is very difficult to say beforehand whether an algorithm of this type will be able to solve a practical problem in a reasonable time. In practice, a branch-and-bound algorithm can be truncated if it exceeds its time limit without proving optimality of the best solution found. Such truncated optimization algorithm can often be of practical use and provide a good approximate solution, together with a corresponding optimality gap.

Another generic solution technique in combinatorial optimization is *dynamic programming* (Nemhauser and Wolsey, 1988). Dynamic programming provides a framework for decomposing certain optimization problems into a nested family of subproblems, that suggests a recursive approach for solving the original problem from the solutions of the subproblems (Bellman and Dreyfus, 1962; Hadley, 1964; Denardo, 1982).

Practitioners often resort to approximation algorithms and simulations to solve their problems. The formal reason is that often the problem at hand is $\mathcal{NP}$-*hard* and thus optimization is likely to be cumbersome and time consuming. Approximation algorithms and simulations, on the other hand, are expected to produce a reasonable solution fast at a low development cost, high flexibility and ease of future refinement. There are, probably, only a couple of problems types on which such expectations are likely to be frustrated. Notable examples are constraint satisfaction problems (Tsang, 1993), where the solution space is sparse and finding a feasible solution is a problem in itself. In this case an optimization technique can be a better option, both in terms of efficiency and development cost, see Chapter 3.

The most respected class of approximation algorithms includes algorithms which incorporate some sort of lower bounding technique. Lower bounds provide a posteriori information about the optimality gap and may also be used within an algorithm to guide it to solutions with good upper bounds. The knowledge of the optimality gap may be useful when, for example, an OR analyst is consulted on an important decision problem and an approximation algorithm comes up with an unacceptably costly solution. It can then be crucial to know whether this high cost is intrinsic to the problem or whether it is likely to be improved by further optimization. The use of lower bounds in constructing good feasible solutions to the original problem can be manifold. In case of an ILP, a feasible solution can be obtained from a relaxation of the ILP by iterative fixing the variables, see for example (Caprara et al., 1997), or by rounding the solution. Sometimes an optimization algorithm is applied within an approximation algorithm to solve a certain subproblem (Kolen, 1997), see Chapter 3.

Another big group of heuristic algorithms is called *local search* (Aarts and Lenstra, 1997). These algorithms are classified according to the way in which they work. They all start with an initial solution or a group of solutions and perform iteratively small modifications of solutions in search for an improvement. A typical example of this technique is the simplex method for linear programming (Chvátal, 1983). As far as the application of popular local search techniques such as tabu search, simulated annealing, genetic algorithms, and neural networks, to combinatorial optimization is concerned, their differences are much smaller than it may look at first glance, see Chapter 3. The choice of a particular technique essentially comes to a matter of taste, but what really matters for its successful performance is the careful use of the structure and the properties of the problem at hand in interpreting such concepts as the neighborhood function and the search strategy, see Chapter 3.

Simulation techniques (Law and Kelton, 1991) are relatively unpopular in combinatorial optimization. I believe they are especially useful for model validation, see Section 1.1, and for nasty types of practical problems in which solution evaluation is costly, see Chapter 4. Simulations are still the state of the art for solving practical online optimization problems (Ascheuer et al., 1998).

Of course, any attempt of a general presentation of solution techniques is bound to be incomplete. The story in this section is no exception in this respect. For example, it failed to mention algorithms for specially structured problems, like network and graph

based algorithms (Ahuja et al., 1993; Bertsekas, 1998), or newly emerging techniques, like potential reduction (Warners et al., 1997a) and semidefinite programming (Alizadeh, 1995; Goemans, 1997).

There are plenty of software tools, implementing combinatorial optimization algorithms. The best references on this subject are the following WWW-pages:

- Michael Tricks's Operations Research Page, available at `http://mat.gsia.cmu.edu/resource.html`;
- NEOS Guide to Optimization, available at `http://www-fp.mcs.anl.gov/otc/Guide/SoftwareGuide/index.html`.

Besides good references to the existing software, these pages also contain plenty of useful tips and tricks concerning the practical use of the software.

1.3 Abstracts

The purpose of this section is to give the reader the flavor of the things to come and to encourage (or to discourage) him or her to read the full presentation of the case studies in the subsequent sections.

Case 1: Production scheduling

A complex manufacturing process at Van Geel B.V. in Boxtel is modeled as a generalized job shop scheduling problem. In this problem we are given a set of jobs to be processed by a set of machines. A job consists of a sequence of operations. Each operation is assigned to a certain machine and its processing time is known. The problem then is to assign starting times to operations so that precedence, release and changeover constraints are respected. Moreover, machines are operated in shifts, which means that each of them has its own operating hours. The criterion is to minimize the maximum lateness, i.e., the difference between the due date of a job and its actual completion time.

A tabu search algorithm is used for this problem. The algorithm is designed to provide the planner with a good solution within several minutes of computing and with a near-optimal solution if several hours of computing are allowed. An implementation of the algorithm is incorporated in a decision support system used by the company's planning department.

This was my first full scale operations research consultancy project. The main challenges of this project, such as model building and validation and algorithm selection, also became the main motivation for writing this thesis. The next step in understanding the empirical rules of algorithm selection is given in the following study.

Case 2: Frequency assignment

Frequency assignment problems occur when a network of radio links has to be established. Each radio link has to be assigned an operating frequency from a set of available frequencies.

A frequency assignment has to satisfy certain interference limiting restrictions defined on the pairs of links. The amount of spectrum tied up by an assignment is to be minimized.

This project is a laboratory test of the algorithmic techniques. It provides a great opportunity for a direct comparison of widely used algorithmic approaches and to derive empirical criteria for algorithm selection.

Algorithms for frequency assignment problems. This is an overview of contributions to the CALMA project coauthored with Karen Aardal, Cor Hurkens, and Jan Karel Lenstra. The goal of the project was to assess the relevance of different algorithmic techniques for a selected military application. Criteria were to be worked out for algorithm selection and evaluation.

These issues were addressed by a consortium consisting of research groups from Delft, Eindhoven, London, Maastricht, Norwich, and Toulouse. The participants developed optimization algorithms such as branch-and-cut and constraint satisfaction, and approximation techniques like simulated annealing, tabu search, variable-depth search, genetic algorithms, potential reduction, and partial constraint satisfaction. These algorithms were compared on a set of real-life and random instances.

Local search algorithms for frequency assignment problem. This work coauthored with Cor Hurkens and Jan Karel Lenstra is our contribution to the European CALMA (Combinatorial ALgorithms for Military Applications) project. We introduced a new model for the radio link frequency assignment problem. Our part of the project was to develop and to test several local search algorithms, including simulated annealing, tabu search and variable-depth search, with the emphasis on the design of efficient neighborhood functions.

The algorithms were tested on a number of real-life and random instances. We derived lower bounds based on a nonlinear relaxation of the problem for some of these instances.

Case 3: Statistical disclosure control

A microdata file contains records with information collected by a statistical office from individual respondents, typically by means of a survey. Such files are intended for public release on the condition of protecting confidentiality of the respondents, i.e., preventing identification of the respondents in the microdata file. In practice, certain data operations are applied to reduce the risk of identification in the file to a reasonable level. Statistical disclosure control is the problem of the optimal application of such operations at a minimum information loss.

In this work coauthored with Cor Hurkens we propose a set covering model. We discuss several algorithms for this formulation, based on Lagrangean relaxation and local search.

Despite our efforts, I consider this project to be a failure. We failed to develop an adequate model, because we were unable to validate it properly. Even for the chosen model we developed inefficient, cumbersome algorithms which appeared to be of little help

in a decision support tool for statistical disclosure control. The lessons I have learned from this project are summarized in the conclusions to Chapter 4.

Case 4: Industrial cutting

DUMO N.V. in Goirle is a manufacturer of furniture foam components. The main production operation of the company is to cut huge rectangular foam blocks into smaller ones, which are used for end-products at the later stages. We model this process as the three-dimensional cutting stock problem. In this problem, the aim is to cut a set of rectangular blocks of known size and demand from a set of standard blocks with a minimum waste.

In our joint work with Cor Hurkens we develop an incomplete optimization algorithm based on column generation. Our decision support tool solves large scale instances of the problem in at most 15 minutes. This tool has proven to be useful in the existing production process. Numerous online tests indicate consistent improvements with respect to manually generated solutions.

Chapter 2

Case study: Production scheduling

2.1 Introduction

Van Geel Metal B.V. is a manufacturer of metal constructions mainly used in interior design. Among the most important product types are suspended ceilings and partitions. The company also performs cladding and casing of various materials used in interior design. These products are manufactured at a factory in Boxtel, the Netherlands.

Production in the factory is organized on customer orders. Each order requires a series of development and production activities. Sections 2.1.1 and 2.1.2 describe the production and the planning processes of the company, respectively.

The main motivation of our consultancy efforts was to develop a mathematical model of the production process at Van Geel Metal with the goal to come up with useful algorithms and software tools to assist planners in making good production scheduling decisions. Sections 2.2–2.4 present our approach, discussing in turn a mathematical model, a mathematical solution technique and its implementation. In Section 2.5 we throw a critical glance at the results of the project, and at the advantages and the shortcomings of the proposed solution approach.

2.1.1 Production process

Typically, a product at Van Geel Metal requires the execution of a sequence of operations. The most important operations are:

- **perforating and cutting:** a metal plate is cut and perforated with an appropriate pattern;
- **long side bending:** the long side of a plate is bent;
- **short side bending:** the short side of a plate is bent;
- **powdering:** powdering and enameling of a plate in a desired color;
- **punching:** punching a hole or a ridge in a metal plate;
- **welding:** a simple welding operation;

- **packing:** packing a product in foil or paper.

For a complete list of operations see Van Hoesel et al. (1993). Usually, a product requires only some of these operations. For example, an element of a suspended ceiling may require cutting, perforating, side bending, powdering and packing. The duration of each operation may vary from a couple of minutes to days. A typical operation can be performed by a number of machines. However, each machine has its own specific features and in the most cases there is a unique machine that suits the job better than the others.

Machines are operated in shifts at Van Geel Metal. The processing of a job can be terminated by a machine at the end of a day and be resumed the next day without any extra loss of time. All machines at the factory can process only one job at a time. Although the set and the order of operations required for a product differ significantly from one product to another, there is a relatively small set of machines that are used often. When two operations succeed one another on a machine, this machine often requires adjustment of its setting between the operations. We call such an adjustment a *changeover*, and the time needed for it a *changeover time*. Sometimes, it is possible to save on the changeover time by combining operations that require similar machine settings. For example, a perforating machine uses a certain perforation pattern, specific for each operation. Clearly, if two products require the same perforation pattern they do not need a changeover between them. In this way a gain can be achieved by combining similar products. The same consideration holds for most of other machines at the factory.

Subsequent operations of a job usually require processing on different machines and thereby sufficient time for transportation from one machine to another. Rush jobs are different in this respect. Subsequent operations of a rush job may overlap in time, because a part of the job, when completed, is made directly available for the following operation. Jobs may follow different paths on the shop floor. They may visit different machines in a different order. However, the order in which a job visits machines is fixed for each job.

Raw materials are only used at the first stage. There is no complicated assembly involved and the output of one stage constitutes all the input for the subsequent stage. There are no complicated resource constraints at Van Geel Metal. The factory possesses enough stock space and transportation means to satisfy any realistic demand for them.

2.1.2 Organization of production planning

Newly accepted orders are collected in the planning department. Given specifications of a product, engineers of the department determine its technological sequence of production. Information important for our purposes is generated at this stage. The most important data per product is:

- an ordered set of operations to be performed;
- for each operation a machine to which it is assigned;
- production time and changeover time of each operation;
- release and due dates for each order.

It has to be kept in mind though, that all the times in this list are rough estimates, except maybe for the due dates.

When an order is ready for production, it is placed in the pool of ready-to-execute jobs. Once a week the situation on the shop floor is examined and some jobs from this set are released for production. A rough cut capacity check is performed to ensure that the available capacity of the machines is not exceeded by the jobs assigned to them. However, in real life the standard capacity is never enough to perform the work required. Therefore, a planner has either to extend the capacity of some machines or to delay the execution of some jobs. The capacity of a machine can be extended by introducing one or two extra working shifts per day. The goal of a planner is essentially to execute all released jobs before their due dates with the smallest possible extension of capacity. Such a decision is taken on the basis of information aggregated over a week, neglecting possible conflicts of jobs for capacity.

When the target capacity of machines and the set of jobs to be processed in a week have been determined, the rest is entrusted to the operators of machines. They decide about the actual sequence of jobs in production with the goal to use the machine capacities efficiently, i.e. to minimize the changeover time. Leaving the sequencing decisions to the operators of machines certainly encourages the people to be responsible and to employ their creativity in using the expensive machines efficiently. However, there is a clear conflict of interests between the planning and the production departments of the company. Namely, that between executing the jobs on time and maximizing the usage of existing capacity.

The managers of Van Geel Metal see an opportunity to increase productivity and quality of the customer's service in introducing a computer based planning system. Such a system is intended to work with disaggregate information and provide realistic production scenarios at low cost. This system will enable a planner:

- to use the flexibility in capacity effectively;
- to use the capacity efficiently;
- to enable the execution of each order before its due date;
- to react quickly and adequately to unexpected situations on the shop floor;
- to observe overall effects of alternative decisions, and to perform what-if analysis.

Unfortunately, in the new organization the sequencing decisions will be made by a planner, and therefore machine operators will be excluded from the decision making process altogether.

2.2 Model

We will describe the model in the incremental way. We begin with the definition of the standard job scheduling shop problem, our basic model. Then, we gradually add features specific for the production process at Van Geel Metal and eventually summarize our working model.

In order to define the *job shop scheduling problem* consider a set J of jobs, a set M of machines, and a set O of operations. For each operation $u \in O$ there is a job $j_u \in J$ to which it belongs, a machine $m_u \in M$ on which it requires processing, and a processing time p_u. A binary relation '$\rightarrow$' is defined between adjacent operations of a job: if $u \rightarrow v$, then operation u has to be completed before the operation v may start. The problem is to assign a starting time s_u to each operation $u \in O$ so that

$$\max_{u \in O} \quad s_u + p_u \tag{2.1}$$

is minimized subject to

$$\begin{array}{rcll} s_u & \geq & 0 & \forall u \in O, \\ s_u - s_v & \geq & p_v & \forall v, u \in O : \ v \rightarrow u, \\ s_u - s_v & \geq & p_v \ \vee \ s_v - s_u \ \geq \ p_u & \forall v, u \in O : \ m_v = m_u. \end{array} \tag{2.2}$$

Constraints (2.2) are called *availability*, *precedence* and *capacity* constraints, respectively. The capacity constraints ensure that no two operations can be processed simultaneously on the same machine.

A feasible assignment of starting times to operations is called a *schedule*. A schedule is called *left justified* if no operation can be assigned an earlier starting time without changing the processing order of operations. The value of the objective function (1) of a schedule is called the *makespan* of a schedule. We will now describe the *disjunctive graph* model (Roy and Sussman, 1964; Balas, 1969), which will be useful to illustrate our solution techniques. For this purpose consider a graph $G = (O, A, E)$, with a set of nodes O, a set of arcs A and a set of edges E. Each node is assigned a weight equal to the processing time of the corresponding operation. There is an arc (u, v) in A whenever $u \rightarrow v$. Each pair of operations assigned to the same machine generates an edge in E. An example of a disjunctive graph is given in Figure 2.1.

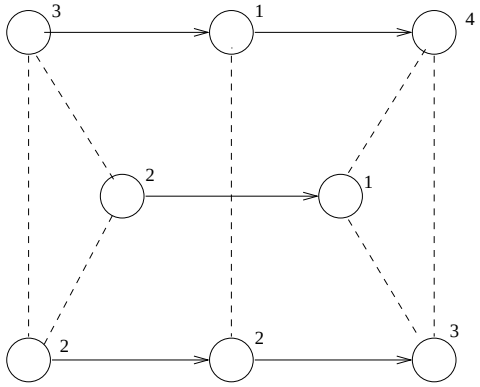


Figure 2.1: Disjunctive graph of a 3-job 3-machine instance.

A processing sequence of operations on each machine is modeled by selecting a direction of every edge in E, so that the resulting graph is acyclic. After such a selection has been made for every edge in E the value of the makespan of the corresponding left justified

schedule is equal to the length of the longest path in G, in which the length of a path is defined as a sum of the weights of nodes on the path. Such a longest path is also called a *critical path*. Then in terms of disjunctive graph model, the problem (2.1), (2.2) is to select a direction for every edge in E, so that the resulting graph is acyclic and the length of its longest path is minimal.

We will now describe a generalization of the standard job shop scheduling model as we used it to model the production process at Van Geel Metal.

Changeover times. Some machines at Van Geel Metal require changeover between two subsequent operations, see Section 2.1.1. A changeover time before an operation v may depend on a whole sequence of operations preceding v on machine m_v. All processing times of operations normally include changeover times in a worst case scenario. For each operation v a better estimate of its processing time can be made if a sequence of operations preceding v on machine m_v is fixed. We call this improved estimate of processing time a *sequence dependent processing time*, and it is denoted by $\hat{p}_v,\ v \in O$.

Waiting times, release and due dates. A waiting time w_{vu} between operations v and u models the slack between the execution of two operations bounded by a precedence constraint. We replace constraints (3) in the model by

$$s_u - s_v \geq \hat{p}_v + w_{vu} \qquad \forall v, u \in O : v \to u. \tag{2.3}$$

The parameters w_{vu} are unrestricted. If w_{vu} is negative for a given pair v and u, then the processing of u may begin before the execution of v is fully completed.

We also introduce a release date $r(\iota)$ and a due date $d(\iota)$ for each job ι, so that the first operation of job ι may not start before $r(\iota)$, and the last operation of ι should finish by time $d(\iota)$. Unlike the former restriction, which is hard, the due date of a job may be violated, although such a scenario is discouraged.

Machine capacities. At Van Geel Metal machines are operated in shifts, i.e. succeeding operational periods of variable starting time and duration. The following parameters are used to model this feature:

- $R(\mu)$ - the starting time of the first shift on machine μ.
- $S(\mu)[l]$ - the starting time of the first shift on machine μ on day l.
- $W(\mu)[l]$ - the duration of the shift on machine μ on day l.

Machines at Van Geel Metal can interrupt processing a job at the end of a shift, and resume processing of the same job at the beginning of the next shift without any extra changeover time. Preemption, i.e. interruption a job's processing by another job, is not allowed. We assume that there is at most one shift per day on a machine. This is a realistic assumption, because even when there are more shifts per day, one shift is directly succeeded by the following one, and therefore they can be modeled as one shift.

We will formulate our model in terms of the completion time function $c(.)$ defined on the set of operations. Consider a schedule and an operation v that starts on day l in this schedule. In order to determine whether this operation can also be completed on day l we calculate $X = \hat{p}_v + s_v - W(m_v)[l] - S(m_v)[l]$. If X is nonpositive, then the completion time of v is given by $c(v) = \hat{p}_v + s_v$; otherwise X is the remainder of the processing time of v, and is carried over to day $l+1$. This procedure is repeated until $X \leq 0$.

We define the completion time of an operation v as follows:

$$\begin{aligned} c(v) \quad &= \quad \{S(m_v)[l^*] + (\hat{p}_v - \sum_{l=k^*}^{l^*-1} W(m_v)[l] + s_v - S(m_v)[k^*])\}, \qquad (2.4) \\ &\text{where} \quad k^* = \max_i \{i | S(m_v)[i] \leq s_v\}, \\ &\text{and} \quad l^* = \min_i \{i | (\hat{p}_v - \sum_{l=k^*}^{i} W(m_v)[l] + s_v - S(m_v)[k^*]) \leq 0\}, \end{aligned}$$

i.e. k^* and l^* are the dates of the start and completion of v, respectively; $c(v)$ is calculated as the starting time of the shift on machine m_v on day l^* plus the remaining processing time of v on that day.

The completion time of a job ι is defined as $c(\iota) = \max_{u \in O,\ j_u = \iota} c(u)$. The difference between the completion time of a job and its due date is called *lateness* of a job. Our objective is to minimize maximum lateness. The problem is to assign a starting time s_u to each operation $u \in O$ so that

$$\max_{\iota \in J} \quad c(\iota) - d(\iota) \qquad (2.5)$$

is minimized subject to

$$\begin{array}{lll} s_u \geq r(j_u) & \forall u \in O, & \\ s_u \geq c(v) + w_{v,u} & \forall v, u \in O: \ v \rightarrow u, & \qquad (2.6) \\ s_u \geq c(v) \ \vee \ s_v \geq c(u) & \forall v, u \in O: \ m_v = m_u, & \end{array}$$

where $c(v)$ is given by (6) for all $v \in O$.

This problem is a generalized job shop scheduling problem with release and due dates, changeover and waiting times, and machine time slots. The optimality criterion is to minimize maximum lateness. For a given optimal value, it is desirable to a find an optimal schedule with a minimum overall changeover time. Therefore, we use the total changeover time as a secondary objective.

2.3 Solution approach

We begin by transforming an instance of the problem (2.5), (2.6) to an instance of an equivalent problem with the makespan as the objective. For this, a dummy operation is appended to each job to make the due dates of all jobs equal, to say D. For each dummy operation u we create a dummy machine m_u with such a shift that if u starts on time $d(j_u)$ on machine m_u it will be completed at time D.

After the foregoing transformation the new problem is to assign a starting time to each operation, including the set of dummy operations, so that

$$\max_{\iota \in J} \quad c(\iota) \tag{2.7}$$

is minimized, under constraints (2.6).

In this section we describe an approximation algorithm for this problem and a post-processing procedure for reducing total changeover time. These algorithms are based on the principle of local search (Aarts and Lenstra, 1997). The general idea of a local search algorithm is to start with an initial solution and iteratively perform small transformations of this solution, called *moves*, in an attempt to find an improvement.

A *neighborhood* of a given solution is defined as a set of solutions to which a given one can be transformed in one iteration. A mapping that specifies a neighborhood of each solution is called a *neighborhood function*. A solution that has no solution with better objective value in its neighborhood is called a *local optimum*. There are several ways to use a neighborhood mechanism for finding good solutions.

Iterative improvement modifies at each step the current solution to a solution from its neighborhood of lower cost. This algorithm stops at the first local optimum it encounters.

Tabu search (Glover, 1989; Glover, 1990) always moves to the best solution in a neighborhood. In this way the cost of solutions generated is not necessarily decreasing. To prevent this algorithm from cycling, usually the reversal of several recently performed moves is prohibited. A stopping criterion has to be defined, for example a maximum number of iterations without improvement.

In the following subsection we describe a procedure for obtaining an initial solution. Then, in Section 2.3.2 two neighborhood functions are defined. We discuss the implementation of a tabu search algorithm and our post-processing procedure in subsequent sections.

2.3.1 Finding an initial solution

We use a greedy algorithm to construct an initial solution. This algorithm starts with an empty schedule and adds one operation at a time. Let $r(u)$ denotes the release time of operation u, i.e. the earliest possible starting time. The algorithm is as follows:

1. Initialize the list of available operations L by including the first operation of each job. L is ordered in nondecreasing values of $r(.)$.
2. Select the first operation v from L and calculate the completion time $c(l)$ of the last operation l scheduled so far on machine m_v (if no operation is scheduled on m_v, then $c(l) = R(m_v)$).
3. For each operation $u \in L$ which is eligible to start at time $\max\{c(l), r(v)\}$ on machine m_v, calculate a reduction of the changeover time $H_u = p_u - \hat{p}_u$.
4. Delete operation $y = \arg\max_{u \in L:\ m_u = m_v,\ r(u) \leq \max\{c(l), r(v)\}} H_u$ from L and start it on machine m_v at time $\max\{c(l), r(y)\}$. Add operation $k \in O$ with $y \rightarrow k$ to L (if it exists) with release time $r(k) = c(y) + w_{yk}$.

5. If L is empty, then stop, otherwise go to step 2.

This procedure generates a left-justified schedule with respect to constraints (2.6).

2.3.2 Neighborhoods

Neighborhood 1

A *swap neighborhood* is one of the most widely used neighborhood structures for scheduling problems. In this neighborhood two operations in a given schedule are swapped to obtain a neighboring solution. A typical swap neighborhood for the standard job shop scheduling problem swaps two adjacent operations on the critical path scheduled on the same machine. This neighborhood possesses some important properties for the job shop scheduling problem. Consider a feasible schedule and let G be its disjunctive graph representation. The first property of this neighborhood is that the graph G' after such a swap will remain acyclic and therefore there exists a feasible schedule corresponding to G' (Vaessens et al., 1996). Moreover, swapping any two operations not on a critical path can never improve the makespan of a schedule, because this swap does not decrease the length of the critical path. The swap neighborhood for the job shop scheduling problem has also the important property of being *weakly optimally connected* (Van Laarhoven et al., 1992), i.e. an optimal solution can be reached from an arbitrary feasible solution using moves from this neighborhood.

To define a swap neighborhood for our problem we will first need to generalize the concept of the critical path in the disjunctive graph model. It would be a straightforward task if our problem allowed for a disjunctive graph representation. Unfortunately, we were not able to find such a representation, but we nonetheless use the term *critical set* as an analog of critical path in our case. The *critical set* of a left-justified schedule is an ordered set of operations. We define the critical set by induction.

- The last operation u of the critical set is chosen such that $c(u) = \max_{v \in O} c(v)$, i.e. its completion time is equal to the makespan of the schedule. Ties are broken arbitrarily.
- Let operation v be in the critical set. The predecessor of v in the critical set is an operation w such that $c(w) = s_v$ and either $w \to v$ or $m_v = m_w$. Ties are broken arbitrarily. If v does not have a predecessor, then v is the first operation in the critical set.

In other words, the critical set is a set of operations that determines the objective value of a schedule, so if one wants to improve a schedule something has to be done with this set. We have chosen to swap two adjacent operations in the critical set if they are scheduled on the same machine. However, there is no guarantee that solutions obtained in this way are feasible. As an example of an infeasible swap consider a 2-job 2-machine instance on Figure 2.2. Jobs A and B in this instance have to be executed on machines M_1 and M_2, which are operated in one (8 hours) and in two (16 hours) shifts per day, respectively. The critical set of the schedule in Figure 2.2 consists of operations A_1 and B_2, because the former operation directly precedes the latter on machine M_1. However, swapping these two operations will result in a feasible schedule.

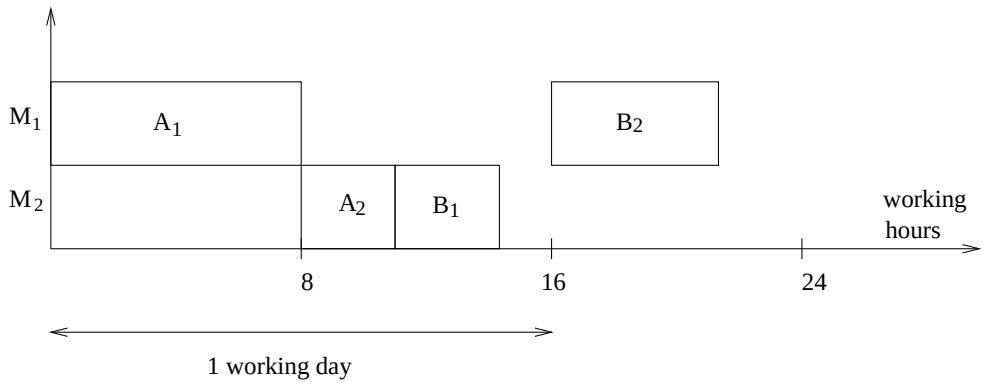


Figure 2.2: A Gantt chart of a schedule for a 2-job 2-machine instance.

Neighborhood 2

Our second neighborhood is also a swap neighborhood. Here, we only consider swaps on machines which require changeover, see Section 2.1.1. Any two operations on such a machine can be swapped in this neighborhood. However, we limit this neighborhood to solutions with the makespan not exceeding the given one.

2.3.3 Local search

Main objective: tabu search

We incorporated the first neighborhood in a tabu search framework. The initial schedule is obtained as described in Section 2.3.1. The algorithm proceeds iteratively by selecting the best solution from the neighborhood. To prevent the algorithm from cycling we prohibit to perform any of the swaps recorded in a tabu list. A so-called aspiration criterion is used to overrule the forbidden status of a swap if it leads to a schedule with the best makespan found so far during the search. When an overall improvement is found all of its neighboring solutions are recorded in a set B. If no improvement is found in I iterations we restart the search from a solution in B. If solution π' is a neighbor of solution π, we call the swap used to obtain π' from π the *generating swap* of π'; $z(\pi)$ denotes the makespan of π. The tabu search with backtracking is as follows:

1. Start with an initial schedule π. Let a tabu list T and a set of active neighbors B be empty; let $M \leftarrow z(\pi)$, and N, J and I be given integer numbers, denoting the size of the tabu list, the maximum number of backtracks and the maximum allowed number of iterations without improvement, respectively. Set the iteration counter i and the backtrack counter j to 0.
2. If $i < I$, go to Step 3. If B is empty or $j \geq J$, then stop. Take the best schedule π from B. $B \leftarrow B \backslash \{\pi\}$, $i \leftarrow 0$, $j \leftarrow j + 1$.
3. Let $\mathcal{N}(\pi)$ be a set of neighbors of π. Select the best schedule $\pi' \in \mathcal{N}(\pi)$. If $z(\pi') < M$, then let $M \leftarrow z(\pi')$, $B \leftarrow \mathcal{N}(\pi) \setminus \{\pi'\}$, $\pi \leftarrow \pi'$, $i \leftarrow 0$ and repeat Step 3. Select the

best schedule $\pi' \in \mathcal{N}(\pi)$ for which the generating swap is not in T; ties are broken arbitrarily. If no solution is selected, then stop.

4. Add the reverse of the generating swap of π' to T. If the size of T exceeds N, delete the earliest inclusion in T. $i \leftarrow i + 1$. Go to Step 2.

Secondary objective: iterative improvement

We apply an iterative improvement algorithm with the second neighborhood to the resulting solution of the tabu search in order to minimize our secondary objective. Solutions generated in the post-processing step may not have a makespan exceeding that of the initial solution of this step, but the total changeover time may reduce.

2.4 Implementation and testing

For the implementation of the algorithms we have chosen the object oriented programming language *C++*. This language is known to produce fast code and it is very convenient when working with complex data. It allows to describe objects by means of data and functions defined on this data. A graphical user interface and an interface with the administrative system of Van Geel Metal was implemented by ORTEC Consultants B.V.

Initial testing was performed on several benchmark instances, including the notorious 10×10 instance of the job shop scheduling problem (Fisher and Thompson, 1963), for which our algorithm found a solution of 936 in less that 5 minutes on a personal computer.

Extensive testing has also been performed on real-life instances provided by Van Geel Metal. A first round of testing gave us a lot of insight in the essence of the problem. In fact the model described in this chapter was shaped after this first test round was performed, with a prototype of our algorithm for the draft of our model. After the feedback was incorporated into the model, we performed a second round of testing. This round made it clear that we needed a procedure for updating the machine capacities in order to reach an acceptable schedule. In Section 2.1.2, we described the planning process and emphasized that the main goal of a planner is to complete all the released jobs before their due dates. In practice, however, the standard capacity of machines is often not sufficient to reach this goal. Therefore, the planner is faced with the difficult problem of locating bottleneck machines and assigning extra capacity to them. In order to assist the planner in this decision we suggested the following procedure:

1. Find a schedule using our algorithm for an instance with the current setting of parameters (initially all machines are available 8 hours per day).
2. If there are no late jobs in the current schedule, then stop. Otherwise determine one or more bottleneck machines lying on the critical path. Introduce extra shifts for these machines at appropriate dates. Stop if there are no bottleneck machines or if no extra shifts can be introduced. Repeat Step 1.

	Initial solution		Final solution	
iteration	makespan	number of late jobs	makespan	number of late jobs
1	7208	8	6769	13
2	6616	7	5801	10
3	6566	6	5939	8
4	6160	6	5610	8
5	5942	3	5323	4
6	5795	2	5144	0

Table 2.1: Example of controlling machine capacities.

Most of our test instances were so tough that even the final solutions contained late jobs. However, it says probably more about the tough reality of operating the factory rather than about the quality of our algorithm.

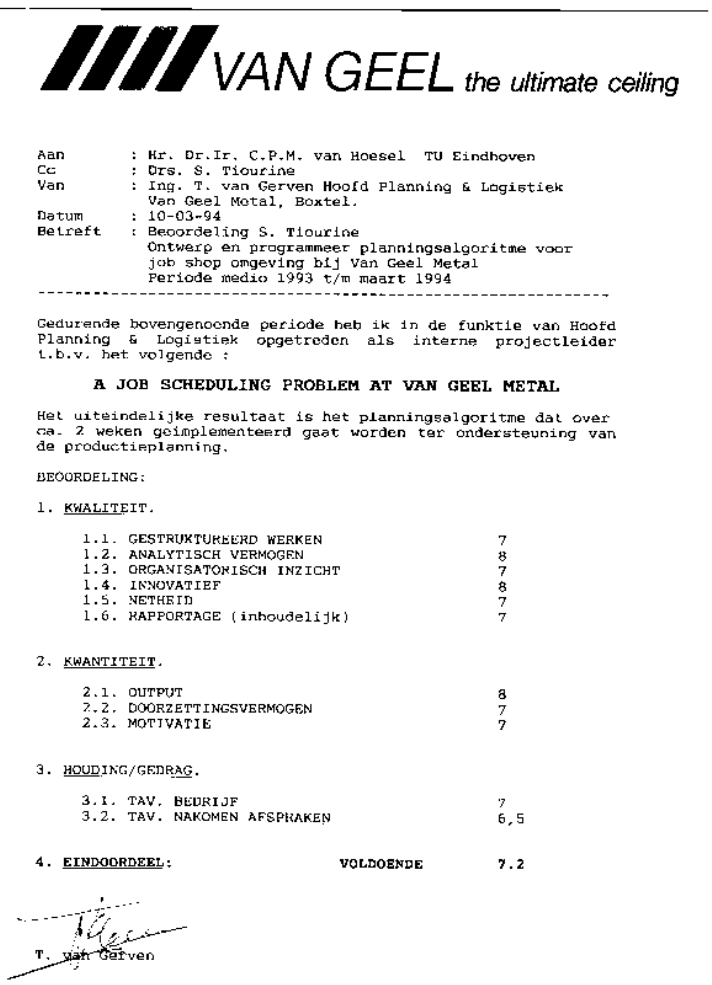
An example of the application of this procedure is given in Table 2.1. In this instance there are 31 machines and 120 jobs, each requiring about 10 operations. Between iterations we introduces one or more extra shifts on the bottleneck machines. The computation time of each iteration was limited to 10 minutes. All in all, some 15 extra working shifts were introduced in this example. This procedure was adopted by the planners at Van Geel Metal and used by them to generate production plans.

2.5 Results and conclusions

I have decided to write this section from two different viewpoints. The first one is the viewpoint I had at the end of the project in January 1994. Another one is from today's perspective, in August 1999.

January 1994. By the end of the project the system was up and running. It was a new toy for everybody in the company. I was in big demand explaining things, showing what can be done with the system. My general impression of the project at the time was that of a moderate success, which was based on the following observations:

1. I had managed to overcome all of the hurdles of the modeling phase and to come up with a model that looked quite realistic to me.
2. I had picked up an algorithm that had a good reputation for the standard job shop scheduling problem and adapted it to the model I had designed for Van Geel Metal. My implementation of this algorithm was running on real-life instances without visible difficulty.
3. The software we produced did not crash and it had a nice interface. My module of the software was responsive to the user's demands, running within its time limits and producing good looking results.
4. I received good grades for my graduation project, see Figure 2.3.



VAN GEEL the ultimate ceiling

Aan : Hr. Dr.Ir. C.P.M. van Hoesel TU Eindhoven
Cc : Drs. S. Tiourine
Van : Ing. T. van Gerven Hoofd Planning & Logistiek
Van Geel Metal, Boxtel.
Datum : 10-03-94
Betreft : Beoordeling S. Tiourine
Ontwerp en programmeer planningsalgoritme voor
job shop omgeving bij Van Geel Metal
Periode medio 1993 t/m maart 1994

Gedurende bovengenoemde periode heb ik in de funktie van Hoofd Planning & Logistiek opgetreden als interne projectleider t.b.v. het volgende :

A JOB SCHEDULING PROBLEM AT VAN GEEL METAL

Het uiteindelijke resultaat is het planningsalgoritme dat over ca. 2 weken geimplementeerd gaat worden ter ondersteuning van de productieplanning.

BEOORDELING:

1. KWALITEIT.

1.1. GESTRUKTUREERD WERKEN	7
1.2. ANALYTISCH VERMOGEN	8
1.3. ORGANISATORISCH INZICHT	7
1.4. INNOVATIEF	8
1.5. NETHEID	7
1.6. RAPPORTAGE (inhoudelijk)	7

2. KWANTITEIT.

2.1. OUTPUT	8
2.2. DOORZETTINGSVERMOGEN	7
2.3. MOTIVATIE	7

3. HOUDING/GEDRAG.

3.1. TAV. BEDRIJF	7
3.2. TAV. NAKOMEN AFSPRAKEN	6,5

4. EINDOORDEEL: VOLDOENDE 7.2

T. van Gerven

Figure 2.3: My grades at Van Geel Metal.

There were some minor irregularities, which I sensed at the time but did not pay much attention to. They were of the following nature:

1. During some of the test runs there were rush orders that should have been delivered already but still had to be produced. My algorithm stopped in a matter of seconds on such instances, reporting an optimal solution. Indeed, a solution generated for such an instance had a critical path composed of operations of one and the same job. An upper bound given by such a schedule coincides with the lower bound on (2.5) equal to the sum of processing times of operations of the critical job, minus possible overlap of its operations, minus the due date of the job. Running my algorithm on such instances did not make much sense, because the schedules produced apart of the rush order were practically random. I saw that these explanations of mine did not mean much to the planners, but I thought that sooner or later they would find out anyway.

2. On one occasion I had to rush to the company because of a reported crash of the system. It appeared that the system was tested on a dummy instance corresponding to an estimated annual production load of the factory. Our calendar manager had crashed on a schedule longer than one year. Such a scenario was indeed difficult to foresee, because the normal planning horizon is only one week.
3. I noticed that machine operators were keen to show that the exact starting times of the operations could never be achieved, which is of course an obvious point. So I tried to argue that the schedule suggested by the system should not be taken literally as the starting times of operations, but it rather should suggest a good production sequence of operations. They went on showing me how exchanging two operations scheduled on the same machine by the system can sometimes significantly reduce the changeover time on that machine. In response I tried to use the what-if analysis implemented in the system to show them the overall effect of the proposed exchange. However, my impression was that my explanations were missing the target once again.

It also became clear at a later stage of the project that no statistics would be available for an evaluation of the system's performance. There is no historical data whatsoever about the actual production times or sequences. This implied that the only possible evaluation of the system's performance was the subjective judgment of the managers and machine operators of the company.

My general conclusion at that point was that in my future research I had to concentrate on two things: on the process of mathematical modeling and on the principles of algorithmic solution of a given mathematical model.

Augustus 1999. As of today, the system appears to have only a limited use. Namely, its what-if analysis block is employed to occasionally compute the effects of selected production scenarios, and sometimes aggregated batches are run to locate the bottlenecks in machine capacities. The system's day-to-day operation has been abandoned due to arguments of the following sort:

- Detailed production schedules are difficult to achieve and to maintain, they are not adaptive to the actual situation on the production floor, and they put a lot of unnecessary pressure on the machine operators.
- The functionality of the system is too narrow and does not always satisfy the needs of the planning department.

I attribute these difficulties primarily to the shortcomings at the modeling and implementation phases of the project. I think that the design and implementation of the algorithms are still surpassing in concept the implementation of the whole system.

From the point of view of today I would suggest a softer approach to this consultancy project. The methodology of OR put forward by Daellenbach et al. (1983) can be used to identify the decision makers and their objectives on the different levels of organization at Van Geel Metal. Such a study can help to resolve the possibly conflicting objectives at the stage of the problem formulation.

I still believe that the type of system described in this chapter may be useful in the current context, but I would develop it in a more systematic way. For example, the unified software development process of Jacobson et al. (1999) provides a framework for a systematic software development which could have been useful in the current project.

Chapter 3

Case study: Frequency assignment

3.1 Introduction

Frequency assignment is a new and rapidly growing area of optimization. Common applications are in mobile telephony (Hao et al., 1998), satellite communications (Ha, 1990), television and radio networks. The problem is to assign a limited number of available frequencies to transmitters located in geographical proximity, in order to minimize electromagnetic interference. The first thorough description of these models and their relations to other well-known optimization problems was given by Hale (1980). Most of these models turn out to be NP-hard, and thus it is highly unlikely to find efficient optimization algorithms for their solution.

This chapter does not pretend to be a thorough review of models and algorithms for frequency assignment, like Koster (1999) for example, neither it is a description of a consultancy project like other case studies in this thesis. Rather, it is a field test of algorithms on a specially structured radio link frequency assignment problem (RLFAP), which stems from a military application (Lanfear, 1989; Hajema et al., 1993). In this problem a network L of radio communications between military units has to be established. Each connection is implemented by a *duplex* pair of links (i, j), i.e., one for up and one for down transmission. Each link i has to be assigned a frequency from its discrete frequency domain D_i, determined by international regulations and hardware requirements. We denote by $\mathcal{L}$ the set of all individual links i in L: $(i, .) \in L$.

The frequencies f_i and f_k of duplex links have to be separated by a fixed distance d_{ik}:

$$|f_i - f_k| = d_{ik}. \tag{3.1}$$

Two communication links i, j operating in geographical proximity cannot be assigned the same frequency, because of the possibility of electro-magnetic interference. Their frequencies have to be more than a distance d_{ij} apart, defined by hardware requirements and signal propagation studies:

$$|f_i - f_j| > d_{ij}. \tag{3.2}$$

We call this an *interference* constraint and the corresponding links are called *interfering* links.

In case a part of the network is already operational, we will prefer not to change the frequencies of the links already in use. Therefore, we introduce so-called *mobility* constraints, by requiring the link i, already in use, to stick to its operational frequency p_i.

Details about the evaluation of the parameters of the model, including a discussion of issues of electromagnetic compatibility, signal propagation, and international frequency regulations, are beyond the scope of this chapter. For these matters, we refer to Lee (1995) and Tuttlebee (1990).

In our modeling approach we consider the radio link frequency assignment problem as a sequence of two problems. We call the first of these the *minimum interference problem* (MIP). Here some of the less important constraints are relaxed and introduced into the objective function with a weight proportional to the importance of the constraint for the functionality of the system. We call such constraints *soft*. The minimum interference problem amounts to finding a frequency assignment that satisfies all *hard* constraints, which are essential for the functionality of the network, and minimizes a sum of costs C_{ij} and M_i for the violation of soft interference and mobility constraints. Denoting by HI and SI the sets of pairs of links bound by hard and soft interference constraints respectively, and by HM and SM the corresponding sets of links for mobility constraints, the problem is to

$$\text{minimize} \sum_{(i,j)\in SI} C_{ij}\delta(|f_i - f_j| \leq d_{ij}) + \sum_{i\in SM} M_i\delta(|f_i - p_i| > 0)$$

subject to

$$\begin{aligned} |f_i - f_j| &> d_{ij} && \forall (i,j) \in HI, \\ |f_i - f_k| &= d_{ik} && \forall (i,k) \in L, \\ f_i &= p_i && \forall i \in HM, \\ f_i &\in D_i && \forall i \in \mathcal{L}, \end{aligned}$$

where $\delta(c)$ is equal to 1 if the condition c is true and 0 otherwise.

The minimum interference problem serves two purposes. First, it determines whether RLFAP has a solution satisfying all soft and hard constraints. Second, in case such a solution does not exist, it gives a solution with the minimum possible level of interference and mobility.

Our secondary objective is to minimize the radio frequency spectrum used. In the *minimum spectrum usage problem* (MSUP) all interference and mobility constraints have to be satisfied, while the number of frequencies used is minimized:

$$\text{minimize } |\cup_{i\in\mathcal{L}} \{f_i\}|$$

subject to

$$\begin{aligned} |f_i - f_j| &> d_{ij} && \forall (i,j) \in SI \cup HI, \\ |f_i - f_k| &= d_{ik} && \forall (i,k) \in L, \\ f_i &= p_i && \forall i \in SM \cup HM, \\ f_i &\in D_i && \forall i \in \mathcal{L}. \end{aligned}$$

Both variants of the frequency assignment problem are NP-hard in the strong sense (Koster, 1999). In the following section we survey the existing approaches to RLFAP. This section is largely based on the results of the CALMA project enriched with some recent results. Our own contribution to the CALMA project is described in Sections 3.3–3.6.

3.2 Survey of algorithms for RLFAP

Radio link frequency assignment problem was investigated in the CALMA project of the EUCLID program. EUCLID, which stands for "EUropean Cooperation on the Long term In Defence," is a research program of West-European Departments of Defence. Three of its members, France, the Netherlands and the United Kingdom, joined in the CALMA project, which had the purpose to investigate the use of "Combinatorial ALgorithms for Military Applications." They chose frequency assignment as the subject of a pilot study, and specified three subprograms: "testing genetic algorithms, testing exact solution techniques, and testing approximate solution techniques" (Hajema et al., 1993). The project was granted to a consortium consisting of six research groups: the Centre d'Etudes et de Recherches de Toulouse (CERT) in France, the Technische Universiteit Delft (TUD), the Technische Universiteit Eindhoven (TUE) and the Universiteit Maastricht (UM) in the Netherlands, and King's College London (KCL) and the University of East Anglia (UEA), Norwich, in the United Kingdom.

In the period from December 1993 to December 1995, each of the groups contributed its expertise to the project. Together they developed and implemented a wide variety of optimization and approximation algorithms. The approximative approaches include neighborhood search methods such as simulated annealing, tabu search and variable-depth search, hyperneighborhood search methods like genetic algorithms, other search methods based on neural networks, incomplete optimization, and potential reduction methods. Such approaches all produce solutions that are hoped to be close to the optimum but that, in the present context, have no a priori quality guarantee. The participating groups also developed techniques for finding lower bounds on the optimum, based on linear programming, graph coloring, constraint satisfaction, and 0–1 quadratic programming. Lower bounds are used as a yardstick in two settings: measuring the quality of upper bounds found by approximation algorithms, and curtailing the search in enumerative optimization algorithms. Finally, all of these approaches were tested and compared on a set of real-life instances.

We used three sets of test instances. One set of eleven real-life instances was provided by CELAR, the Centre d'ELectronique de l'ARmement in France. They range in size between 200 and 916 links, and the values of the interference and mobility costs vary

widely. Among these instances, six are instances of the minimum spectrum usage problem, i.e. there exists a feasible solution to minimum interference problem of zero cost.

A second and a third set of instances were made available by the group at TUD. They were randomly generated, but preserve the structure and main characteristics of the CELAR instances. The second set of fourteen instances was released at the end of the project, and the groups were given one week to report their results. We refer to them as the *surprise* instances, of which six are instances of MSUP. The third set contains seven instances of MSUP and three instances of MIP.

Our test instances have some special structure. For each duplex pair of links (i, k), d_{ik} is equal to a constant $\bar{d}$ ($\bar{d} = 238$ for the CELAR instances), $D_i = D_j$, and for each $f \in D_i$ there is a unique $f' \in D_i$ such that $|f - f'| = \bar{d}$.

3.2.1 Approximation and upper bounds

Approximation algorithms seek to obtain good feasible solutions in a reasonable amount of time. They provide upper bounds on the minimum solution value. Most of the methods to be discussed below apply some form of neighborhood or hyperneighborhood search; one of these uses a representation related to neural networks. We will also discuss an incomplete optimization method, based on truncated tree search, and a potential reduction method, which applies an interior point algorithm and rounding techniques to a binary quadratic formulation of the problem.

Local search. The general idea of local search is to start with an initial solution and iteratively perform small transformations of the solution in an attempt to improve the objective value. The *neighborhood* of a solution is defined as the set of all solutions to which it can be transformed in one iteration or *move*. A mapping that specifies a neighborhood for each solution is called a *neighborhood function*.

A *search strategy* specifies the way in which at each move a solution from a neighborhood is selected. The basic *iterative improvement* strategy transforms the current solution into a neighboring solution of lower cost, and stops when no better neighbor exists. It can easily be trapped in a *local optimum* of poor quality. Search strategies that are intended to overcome this deficiency include simulated annealing, tabu search, and variable-depth search.

Various hybrid forms of algorithms have been proposed, which combine local search with constructive, enumerative or iterative techniques. There exist constructive rules that apply local search to partial solutions, combinations of local search with partial enumeration or backtracking, and nested forms of local search. In the latter case, local search is applied at several levels. For example, a neighbor obtained at one level is subjected to local search at a second level before the search at the first level is resumed.

For a comprehensive discussion of local search techniques and their applications to problems in combinatorial optimization, we refer the reader to Aarts and Lenstra (1997). We will now describe the various local search approaches taken in the CALMA project.

The participants took quite different approaches. TUE emphasized the development of neighborhood functions, which were subsequently used in any of the search strategies. In this way the underlying mechanism of traversing the solution space could be tuned, and also the various strategies could be compared. In contrast, CERT and KCL both considered a plain local search technique and used little additional information to adapt it to the problem at hand.

Simulated annealing. Simulated annealing moves from a solution to a random neighbor. An improvement is always accepted. A deterioration is accepted with a certain probability: Given a solution of value z, the probability that a neighbor of value $z' > z$ is accepted is usually given by $\exp((z-z')/T)$, where T is a control parameter that decreases during the run. The algorithm stops when T reaches a termination value.

CERT. An approach proposed by Bourret (1995) uses a binary solution representation. Each frequency–link combination gets a 0–1 decision variable x_{if} with $x_{if} = 1$ if link i is assigned frequency f and $x_{if} = 0$ otherwise. Exactly one such variable is turned on for each link:

$$\sum_{f \in D_i} x_{if} = 1 \text{ for each } i \in \mathcal{L}. \tag{3.3}$$

All other constraints of the problems are relaxed and their violation is penalized in the objective function. Indicator parameters are introduced to model the violation of the interference and mobility constraints:

$$w_{ij}^{fg} = \begin{cases} 0, & \text{if frequency } f \text{ of link } i \text{ interferes with frequency } g \text{ of link } j, \\ 1, & \text{otherwise,} \end{cases}$$

for $f \neq g$, and $w_{ii}^{ff} = 1$ if $f = p_i$, $w_{ii}^{ff} = 0$ otherwise. A *feasibility function* is then defined by

$$\Phi(x) = \sum_{i,j \in \mathcal{L}} \sum_{f \in D_i, g \in D_j} (1 - w_{ij}^{fg}) x_{if} x_{jg}. \tag{3.4}$$

This function attains its minima in the feasible assignments. For the minimum interference problem a relaxation is obtained by multiplying each term in (3.4) corresponding to a soft constraint in the original problem by a cost coefficient, and each term corresponding to a hard constraint by a very large number, for example the sum of all cost coefficients of the problem plus one, and minimizing the resulting function subject to (3.3). For the minimum spectrum usage problem an *order preserving function* Ω is defined, a surrogate objective function which is intended to increase when the true objective function increases. It is defined by

$$\Omega(x) = \frac{1}{2} \sum_{f \in D, g \in D, f \neq g} \sum_{i \in \mathcal{L}} x_{if} \sum_{j \in \mathcal{L}} x_{jg}.$$

A weighted sum of the feasibility and the order preserving functions is then minimized in this approximative approach.

For both problem types, a move in a neighborhood corresponds to changing the frequency of a single link. The chosen implementation has substantial running time and memory requirements. In order to cope with these difficulties, the author proposed to decompose the test instances into small subproblems and to treat these independently. In spite of various attempts to reduce the frequency domains and to discard links from the problem on heuristic grounds, the approach produced modest results.

TUE. The simulated annealing algorithms of Tiourine et al. (1999) (see also Sections 3.3–3.6) use problem representations and neighborhood functions that are specific to the problem at hand. A graph representation of the problem is based on its similarity to certain graph coloring problems (Roberts, 1989). The *interference graph* $G = (\mathcal{L}, E)$ on the set $\mathcal{L}$ on links has an edge $\{i, j\} \in E$ if and only if i and j interfere. For a list τ of nonnegative integers, a *τ-coloring* is a frequency assignment satisfying $f_i \in D_i$ for all $i \in \mathcal{L}$ and $|f_i - f_j| \notin \tau$ for all $\{i, j\} \in E$. Given a list of admissible frequencies for each link, a solution corresponds to a selection of one frequency from each list that respects the restrictions with adjacent nodes. This representation is used to maintain *arc consistency* (Tsang, 1993), that is, consistency of the domains of adjacent nodes with respect to the current assignment throughout the search.

The authors developed different neighborhood functions and tested these using various search strategies. Because of the special structure of the test instances, they considered duplex pairs of links as atomic objects. In the minimum interference problem, a solution y is a neighbor of a solution x if x can be transformed into y by changing the frequency of a link with a non-zero contribution to the cost of x. This neighborhood is shown to be connected in the sense that, starting from an arbitrary solution and using moves of this type, an optimal solution can always be reached. Computational experiments were encouraging on small and medium-size problems, but the performance on larger instances was less satisfactory.

In the minimum spectrum usage problem, a neighbor of a solution x is obtained by removing a random seed frequency from the set of used frequencies. The links that were assigned this frequency are reallocated by a heuristic procedure. It first tries to use frequencies already used in x. If a link i cannot be assigned such a frequency without violating the interference constraints, forward probing is used to check if reallocation of the links interfering with i may resolve this infeasibility. Experiments with this rather wild neighborhood produced good results.

The implementation of these simulated annealing algorithms uses the cooling schedule proposed in (Van Laarhoven et al., 1992). For each value of the control parameter T, a number of trials is performed, equal to the size of the largest neighborhood. T is modified by the rule $T \leftarrow T/(1 + [T \ln(1 + \Delta)/3\sigma)])$, where Δ controls the decrement rate of T and σ is the standard deviation of the solutions values generated for the current value of T.

Tabu search. Tabu search always moves to the best neighbor. In this way the cost of the solutions generated is not necessarily decreasing. To prevent the method from cycling, several recently visited solutions or the reversals of several recently performed moves are

excluded or "put on the tabu list." A stopping criterion has to be defined, for example a maximum number of iterations without improvement.

KCL. Bouju et al. (1994) describe a tabu search algorithm that changes the frequency of a single link at each move. The size of the neighborhood is restricted to a certain percentage of the links that have the largest contribution to the total interference cost. When applied to the minimum interference problem, the algorithm performed quite poorly. For the minimum spectrum usage problem, they developed an extended algorithm that dynamically updates the frequency domains. Initially, each domain contains two frequencies only and the algorithm searches for an interference-free assignment. If no such solution is found, the frequency domains are extended and the search is repeated. The frequencies are added one by one in the order determined by the number of different frequency domains in which they occur. The approach shows encouraging results.

TUE. The tabu search algorithms of Tiourine et al. (1999) (see also Sections 3.3–3.6) use the same representation and neighborhoods as their simulated annealing algorithms. They apply the backtracking mechanism of Nowicki and Smutnicki (1996) to restart the search from a neighbor of the best solution found when a stopping criterion is met. The moves for the minimum interference problem and the seed frequencies for the minimum spectrum usage problem are put on the tabu list.

This algorithm worked well for the minimum spectrum usage problem. Its poor performance for the minimum interference problem is attributed to the large size of the neighborhoods and the peculiar cost structure of the problem.

Variable-depth search. Variable-depth search was introduced as a highly problem-specific exchange algorithm for uniform graph partitioning (Kernighan and Lin, 1970) and later for the traveling salesman problem (Lin and Kernighan, 1973). In the latter paper the authors give guidelines for possible extensions of the algorithm to other problems. In short, the method works as follows. Starting from an initial solution, it makes a sequence of small greedy moves. This process can be seen as the repeated application of some neighborhood function. It has to be ensured, though, that the moves within the same sequence are not reversed. Improving solutions encountered in such a sequence are registered. Typically the sequence is terminated when it becomes too long or when no gain is expected from further moves. The next iteration starts from the best solution found in the sequence.

TUE. Tiourine et al. (1999) (see also Sections 3.3–3.6) describe two algorithms of this type. Their algorithm for the minimum interference problem starts with a solution that is locally optimal with respect to moves that change the frequency of a single link. It selects a random link with a probability proportional to the cost incurred by the link. The link is assigned a random frequency from its domain. All links interfering with the new assignment are put on a list. A link is then drawn randomly from the list, it is assigned the locally best frequency from its domain, and it is in turn replaced on the list by its interfering links. No link may reenter the list during the same iteration. An iteration is terminated if an improvement is found, or otherwise randomly with a probability proportional to the deterioration of the solution value and the duration of an iteration. This strategy produced

very good results, and even its average running time of 1.5 hours was quite competitive.

For the minimum spectrum usage problem, an iteration of variable-depth search starts by selecting a random used frequency. Each link that is assigned that frequency gets another used frequency from its domain. Infeasibilities resulting from this reassignment are resolved using a variant of the variable-depth search algorithm for the minimum interference problem. This algorithm performed well.

Genetic algorithms. Genetic algorithms also apply local search but now with *hyper-neighborhoods*, where a *set* of solutions is transformed into a new set of solutions. These methods usually mimic mechanisms from evolution theory and typically encode solutions as bitstrings. Starting from a *population* of *parent* solutions, at each iteration its *offspring* is determined by applying *genetic operators*. The binary *crossover* operator aims at propagating the characteristics of good solutions from one generation to the next. The unary *mutation* operator is randomly applied to offspring to ensure a diversity of solutions in the population. It is hoped that the entire process will evolve towards better solutions.

UEA (Chardaire et al., 1995; Rayward-Smith et al., 1995). An early implementation at UEA used GAmeter, their toolkit for genetic algorithms. It applies textbook binary representations, crossover and mutation operators, and produced quite poor results. The introduction of problem-specific operators and data structures significantly improved the performance. One such crossover operator tries to propagate constraints satisfied by parents down to their offspring, another is designed to preserve spectrum usage. Specialized mutation operators implement a variety of ideas that boil down to the application of standard local search to the offspring, so that the overall procedure can be viewed as bilevel search. These refinements led to good results for some of the test instances, but the overall solution quality is not uniform.

UM. The problem-specific and highly tuned genetic algorithm of Kolen (1997) appeared to be one of the most effective approximation techniques for the minimum interference problem. Doing away with the traditional binary representation, random mutations and crossovers, he interpreted the principles underlying genetic algorithms at a conceptual level and developed sophisticated subroutines for the mutation and the crossover operators.

Solutions are represented by vectors of frequencies. At each iteration a new population of a fixed size is constructed by applying a crossover operator to pairs of solutions. A crossover produces one new solution, which is subjected to a mutation operator to ensure its local optimality. Each solution in the parent population is selected in turn for the crossover; its mate is determined randomly with a probability inversely proportional to its value.

The mutation operator applies iterative improvement, changing the frequency of a single link at each move. The crossover is somewhat more complex. For a pair of parent solutions in the population, its offspring is a solution to the original instance of the problem with the domain of each link restricted to the frequencies assigned to it in one of the parent solutions. The algorithm that competed in the CALMA project (Rayward-Smith et al., 1995) used a constructive heuristic for this problem. A later version uses a cutting plane

algorithm; see Section 5.1 (Kolen, 1997).

This approach produced excellent results. The refinement that is to be discussed in Section 5.1 has much lower running times.

Neural networks. Abstracted from their biological background, neural networks represent *connectionist* models of computation. Computations are performed by a network of richly interconnected processors, each performing a relatively simple task. Each processor receives signals from its neighboring processors and calculates a certain response, which in turn is propagated through the network. The computations are performed in parallel and eventually the network should stabilize in some state. The network is *trained* by adjusting the weights of the interconnections.

KCL. KCL obtained promising results using GENET, a generic connectionist tool which simulates neural networks on a sequential computer (Tsang and Wang, 1992). Originally developed for constraint satisfaction problems, GENET was adapted by Vom Scheidt (1995) to the problem at hand.

An instance of the minimum interference problem is represented by a network, each link corresponding to a cluster of nodes, with one node for each frequency in its domain and with an edge between two nodes when the frequencies in question interfere. The mobility and interference costs define weights of the nodes and edges, respectively. A complete frequency assignment now corresponds to a subset of nodes, one from each cluster; its cost is the sum of the weights over the selected nodes and the edges induced by them. The global constraint that exactly one node must be selected from – or be *active* in – each cluster is the major departure from classical neural networks.

At each iteration, one node in each cluster is active. Each node calculates its response depending on the weights of the adjacent active nodes and the incident edges. In each cluster, the node with the maximum response becomes active and the other nodes become inactive. When a local maximum is reached, GENET makes it less attractive by decreasing the weights of the active nodes and of the edges between them. The entire process can be viewed as a bilevel search strategy. At one level, iterative improvement modifies the solution by turning nodes on and off. At another level, the training mechanism changes the instance by adjusting the weights. Good results are reported for the smaller instances.

Major modifications were necessary to apply GENET to the minimum spectrum usage problem. The objective function cannot be evaluated locally anymore and has to be calculated externally after each iteration. Details on how the weights are updated to discourage poor local optima are not reported in (Bouju et al., 1994) or (Vom Scheidt, 1995).

Another approach to the minimum spectrum usage problem amounts to solving a sequence of feasibility problems. Initially, each domain is restricted to two frequencies and the minimum interference problem is solved. The domains are then extended and the process is iterated until an interference-free assignment is found. Dimitropoulos (1998) suggests that the domain extension procedure may follow the same tactics as the tabu search implementation of KCL.

Incomplete optimization. Optimization algorithms usually apply some form of tree search; see Section 3. By limiting the search on heuristic grounds, one may gain speed, but at the expense of losing the optimality guarantee of the solution obtained.

The *partial constraint satisfaction* algorithm of CERT (Bensana and Schiex, 1995) is an example of this approach. It seeks to maximize the number of satisfied constraints. Its performance is unsatisfactory.

Potential reduction. Kamath et al. (1990) and Karmarkar et al. (1991) proposed an interior point algorithm for binary feasibility problems. One first formulates the problem as an optimization problem, by dropping the integrality conditions and introducing a non-convex potential function, whose minimizers are feasible solutions to the original problem. An interior point method is then used to obtain approximate solutions to the new problem. This algorithm starts with a point in the interior of the feasible region and generates a sequence of interior points with decreasing objective value. For that purpose, at each iteration a descent direction for the potential function is determined using its quadratic approximation defined on an inscribed ellipsoid in the feasible region around the last generated point. The algorithm proceeds as long as a substantial reduction in the value of the potential function is found in that direction and no feasible integer solution has been generated by a rounding scheme. When a local minimum is encountered, the potential function is modified in some way and the whole process is restarted. Computation times are mainly influenced by the rate of convergence of the interior point method applied and by the density of the Hessian of the function.

TUD (Warners et al., 1997b). A creative adaptation of this technique to frequency assignment problems uses a specially structured binary quadratic formulation. Let $x_{if} = 1$ if link i is assigned frequency f, and $x_{if} = 0$ otherwise. The minimum interference problem is now to

$$\begin{aligned} \text{minimize} \quad & x^T Q x \\ \text{subject to} \quad & \sum_{f \in D_i} x_{if} \geq 1, i \in \mathcal{L}, \\ & x_{if} \in \{0,1\}, i \in \mathcal{L}, f \in D_i, \end{aligned} \tag{3.5}$$

where Q is a matrix representing the interference and mobility costs.

For the minimum spectrum usage problem, the formulation is slightly modified so as to incorporate an upper bound on the number of frequencies used. Variables z_f are introduced, with $z_f = 1$ if frequency f is blocked and $z_f = 0$ otherwise. The objective $x^T Q x$, which contains terms $x_{if} x_{jg}$ for incompatible simultaneous assignments of f to i and of g to j, is extended to include terms $x_{if} z_f$. In this way a solution of zero cost corresponds to an assignment without interference and without use of blocked frequencies. A constraint $\sum_{f \in D} z_f \geq |D| - N$ is added to ensure that no more than N frequencies are used.

Approximate solutions to both stages of RLFAP are now obtained by solving a relaxation of (3.5), in which all integrality constraints are relaxed and the objective function is of the form $x^T Q x - \sum_j \log w_j s_j$, where s_j is the slack of the jth constraint. In contrast to the model resulting from a straightforward implementation of Karmarkar's approach,

these potential functions have sparse Hessians, which allows for the use of sparse matrix techniques and hence facilitates efficient implementations.

A nice feature of the approach is that it generates many integral solutions. These are obtained while rounding the fractional solutions to the relaxed problem to integer solutions with the same value of x^TQx. The availability of many alternative solutions may be useful when secondary objectives have to be taken into account.

The method obtained fairly good results but has substantial memory requirements. This is due to the size of the models, which have up to 32,000 variables. In some cases various preprocessing routines could be applied to reduce the number of variables and large instances could be solved. The running times were also decreased by the use of a gradient method to search for improving directions in the interior point method, thereby neglecting the second order derivatives that are taken into account in the general scheme. This led to a further improvement of the results obtained by TUD on the CALMA test set (Warners, 1998).

The authors extended their approach to a wider class of quadratic binary optimization problems, whose constraint matrices have specific properties (Warners et al., 1997a).

3.2.2 Optimization and lower bounds

Optimization algorithms for frequency assignment problems must apply some form of enumeration of the solution space, using lower bounds on the minimum solution value as well as logical arguments so as to curtail the search process. A lower bound is usually obtained by relaxing the problem, that is, by discarding some of its constraints and solving the remaining, simpler, problem. We will discuss lower bounding techniques based on linear programming, graph coloring, constraint satisfaction, and 0–1 quadratic programming.

Branch-and-bound. If we wish to solve the frequency assignment problem to optimality, we need to use an enumerative algorithm such as branch-and-bound. To be able to solve real-life instances of such a computationally hard problem in this way, it is crucial to provide the algorithm with a very good lower bound. One way to obtain a lower bound is to take an integer linear programming formulation of the problem and to drop the integrality requirements on the variables. The bound provided by this *linear relaxation* of the problem is usually too weak. *Polyhedral lower bounds* are obtained by identifying additional linear inequalities that strengthen the linear relaxation, and in the best case are necessary in the linear description of the convex hull of feasible solutions. If we would know the linear description completely, then we could solve the problem as a linear programming problem, which is computationally easy. Obtaining such a complete description is, however, as hard as solving the problem itself. We therefore settle for certain families of linear inequalities that define high-dimensional faces of the part of the convex hull of solutions where we expect to find an optimal solution. Since the formulation should not grow too big, we add inequalities only if they are violated by the current fractional solution. An algorithm that at each iteration adds violated linear inequalities to the current formulation and resolves the linear programming problem based on the extended formulation is called a *cutting*

plane algorithm. In *branch-and-cut* we apply a cutting plane algorithm in every node of the tree.

Other lower bounds can be based on combinatorial arguments. Branch-and-bound algorithms generally benefit from preprocessing techniques that reduce the size of the problem instance and from heuristics that generate feasible solutions in each node of the tree.

TUD, TUE. Aardal et al. (1998) developed an optimization algorithm for the minimum spectrum usage problem based on branch-and-bound, using a *vertex packing* formulation. Information about the structure of an instance is extracted from the *interference graph* $G = (\mathcal{L}, E)$ with a vertex i for each link and an edge $\{i, j\}$ between two links whenever $d_{ij} > 0$; cf. Section 2.1. We assume that the duplex pairs of links are given by $(i, i+1)$ with i odd.

The interference graph formulation of MSUP is a graph coloring problem with the additional restrictions that not all colors are admissible for all vertices, and that certain distance requirements have to be satisfied. This formulation is not convenient to write down as an integer programming problem, but it gives important structural information that we shall use when solving the problem.

A formulation that leads more naturally to a mathematical programming formulation is associated with an extended *graph of variables* $G' = (V', E')$. There is a vertex $v_{if} \in V'$ for each combination of a link $i \in \mathcal{L}$ and a frequency $f \in D_i$. There is an edge $\{v_{if}, v_{jg}\} \in E'$ if $|f - g| \leq d_{ij}$. An optimal solution is a subset $S \subset V'$ such that for every link $i \in \mathcal{L}$ exactly one vertex v_{if} belongs to S, such that S forms a *vertex packing* or *independent set* in G' (i.e., no two vertices of S are adjacent), and such that S is of minimum cardinality.

Let $x_{if} = 1$ if link i is assigned frequency f and $x_{if} = 0$ otherwise. Let $y_f = 1$ if frequency $f \in D$ is used and $y_f = 0$ otherwise. The vertex packing formulation of MSUP is now as follows:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{f \in D} y_f \\
\text{subject to} \quad & \sum_{f \in D_i} x_{if} = 1 && \text{for all } i \in \mathcal{L}, \\
& x_{if} + x_{jg} \leq 1 && \text{for all } i, j \in \mathcal{L} : \ |f - g| \leq d_{ij}, \\
& x_{if} + x_{jg} \leq 1 && \text{for all } i, j \in \mathcal{L} : \ j = i + 1, \ i \text{ odd}, \ |f - g| \neq \bar{d}, \\
& x_{if} \leq y_f && \text{for all } i \in \mathcal{L}, f \in D_i, \\
& x_{if} \in \{0, 1\} && \text{for all } i \in \mathcal{L}, f \in D_i, \\
& y_f \in \{0, 1\} && \text{for all } f \in D.
\end{aligned}
\tag{3.6}
$$

The constraints (3.6) can be converted to vertex packing constraints by using the complement $\bar{y}_f = 1 - y_f$ of the variables y_f. This substitution reformulates (3.6) as

$$x_{if} + \bar{y}_f \leq 1 \qquad \text{for all } i \in \mathcal{L}, f \in D_i. \tag{3.7}$$

Clearly, one could extend G' by adding vertices for the frequencies corresponding to the variables $\bar{y}_f$ for $f \in D$, and edges corresponding to constraints (3.7).

Since the domains D_i are large, the above formulation contains many variables. When applying branch-and-bound, the size of the instances and the strength of the linear relaxation of the formulation are possibly the two most crucial issues.

Preprocessing. The number of variables in the CELAR instances varies between approximately 8,000 and 32,000. In order to reduce the size of the instances one can apply various preprocessing techniques. One obvious way of reducing the number of variables is to make use of the restriction that, for i odd, if link i is assigned the frequency $f \in D_i$, link $i+1$ has to be assigned the unique frequency $f' \in D_i$ for which $|f - f'| = \bar{d}$. We can therefore replace all variables $x_{i+1,f}$, i odd, by variables $x_{if'}$. This substitution reduces the number of variables by a factor two. We refer to Aardal et al. (1998) for various other preprocessing techniques.

Polyhedral lower bounds. The classes of inequalities used are all variants of the general class of *clique inequalities*. A clique is a complete subgraph of an undirected graph. Note that the size of a maximum clique in the interference graph G provides a lower bound on the minimum number of frequencies that need to be used. Clique inequalities have the form $\sum_{i \in C} z_i \leq 1$, where the variables z_i are associated with the vertices of the graph under consideration and where C is a clique of that graph. A clique inequality is known to define a facet of the vertex packing polytope if and only if the clique in question is maximal (Padberg, 1973). Given the sizes of G' and of its maximal cliques, it is not realistic to derive clique inequalities from G'. We observe, however, that a clique in the much smaller interference graph translates into a clique in the graph of variables, and therefore search for maximal cliques in G. We refer to Aardal et al. (1998) for further details on the identification of violated clique inequalities and on the implementation of the branch-and-cut algorithm.

Lower bounds from graph coloring. For some of the instances of MSUP the so-called *list chromatic number* dominates the polyhedral bound. Consider G and relax the problem by setting $d_{ij} = 1$ for each edge $\{i, j\} \in E$. This relaxation is known as the *list coloring problem*. The *list chromatic number* of the modified interference graph, i.e., the minimum number of colors needed such that each vertex i receives a color from the set D_i and adjacent vertices receive different colors, is a lower bound on the optimal solution value. Aardal et al. (1998) describe an enumerative algorithm for determining the list chromatic number.

An algorithm incorporating the above ingredients solved all feasible CELAR instances to optimality in a reasonable amount of time (Aardal et al., 1998). The implementation uses MINTO, a software package for integer linear programming. The application of this kind of approach is restricted to problems where a strong linear relaxation is available, which is not yet the case for the minimum interference problem.

Constraint satisfaction. Constraint satisfaction is a specialized backtracking algorithm for integer feasibility problems. In the present context, we wish to find an assignment of frequencies to links respecting a number of constraints. A constraint satisfaction algorithm assigns frequencies to links in an order specified by variable and value selection rules. If a

feasible solution is found, the algorithm terminates. Otherwise, it performs backtracking to the nearest active node with a feasible alternative frequency. It applies consistency enforcing techniques to limit the search and to detect infeasibility of a partial assignment. Two of these techniques, forward checking and arc consistency, appear to be very useful. Given a partial assignment, they seek to reduce the size of the domains of the free links. Forward checking removes all frequencies from these domains that are inconsistent with the partial assignment made. Arc consistency ensures that every frequency in the domain of a free link i has a support in the domain of another free link j if i and j are bound by a constraint. In other words, it will remove a frequency from the domain of i if it cannot be assigned to i to satisfy the constraint with j. For a detailed overview of the field of constraint satisfaction we refer the reader to (Mackworth, 1987) and (Tsang, 1993).

UM. The algorithm of Kolen and Van Hoesel (1995) solves the minimum spectrum usage problem as a sequence of feasibility problems. The number of frequencies used is bounded from above by the value of the best known solution. If this problem is proved to be infeasible, then the best known solution is optimal. Otherwise, the right-hand side of the objective constraint is reduced and the algorithm is resumed. Note that the problem is only strengthened this way. The algorithm does not have to be restarted from scratch, since all nodes fathomed in the previous run will retain their status.

The algorithm uses the same preprocessing step as the polyhedral methods. The variable selection rule is determined after preprocessing. For that purpose, the variables are removed one by one from the problem in such a way that at each step the variable involved in the least number of constraints is removed; the variable selection order is the reverse of the order of removal. The value selection rule takes a random frequency already used in the partial assignment, or otherwise the frequency present in most of the domains of the free links.

The authors also proposed to curtail the search tree by a lower bound based on the chromatic number of the underlying interference graph, thereby stressing the similarity of their approach to depth-first branch-and-bound. They succeeded in proving optimality of solutions for all benchmarks of the minimum spectrum usage problem, albeit at the expense of considerable running times.

A 0–1 quadratic programming relaxation. To obtain a lower bound for the minimum interference problem, TUE considered a relaxation that, instead of assigning a frequency to each link, decides whether or not to adhere to its preassigned frequency. For this relaxation it is easy to define an objective function that covers most of the incurred costs and is subject to integrality constraints only. Denote the interference cost of a pair of links $\{i, j\}$ by K_{ij} if i and j are set to their preassigned frequencies:

$$K_{ij} = \begin{cases} c_{ij}, & \text{if } |p_i - p_j| \leq d_{ij}, \\ 0, & \text{otherwise,} \end{cases}$$

and by K_{ifj} if i is set to f while j is set to its preassigned frequency:

$$K_{ifj} = \begin{cases} c_{ij}, & \text{if } |f - p_j| \leq d_{ij}, \\ 0, & \text{otherwise.} \end{cases}$$

Let the decision variable x_i be equal to 1 if link i is set to its preassigned frequency, and 0 otherwise. The relaxation is then to

$$\text{minimize} \sum_{i,j\in\mathcal{L},i<j} K_{ij}x_ix_j + \sum_{i\in\mathcal{L}} m_i(1-x_i) + \sum_{i\in\mathcal{L}}(1-x_i)(\min_{f\in D_i\setminus\{p_i\}} \sum_{j\in\mathcal{L}} K_{ifj}x_j)$$
$$\text{subject to } x_i \in \{0,1\} \text{ for all } i \in \mathcal{L}.$$

We will interpret the objective function for a link i. If the decision is made not to use the preassigned frequency p_i, in other words if $x_i = 0$, then the second and the third term occur for i. The second term is simply the mobility cost of i. The third term is the minimum cost of interference that occurs between i and all other links that are set to their preassigned frequencies. Suppose now that i is set to its preassigned frequency: $x_i = 1$. The first term gives the cost of interference between i and all other links fixed to their preassigned frequencies. The variable x_i is also considered in the third term, determining the cost of the cheapest alternative frequency for links whose frequencies are different from their preassigned ones. The only interference cost that is not covered by this objective function concerns the interference between links both different from their preassigned ones. Since all cost coefficients in the problem are nonnegative, an optimal solution to the relaxation yields a lower bound to the minimum interference problem.

This approach works well if, first, a significant number of links have a preassigned frequency and at least some of these cannot be changed and, second, if the mobility coefficients m_i are not much smaller than the interference coefficients c_{ij}. For the few instances that satisfy these assumptions, a preprocessing technique and an efficient enumeration scheme incorporated in a branch-and-bound framework were used to obtain lower bounds of a reasonable quality. See Sections 3.3–3.6 for details.

3.2.3 Computational comparison

Tables 1 and 2 present the computational results obtained for the two sets of test instances mentioned in Section 1. For each approach we indicate the quality of the reported solutions and an average running time for the classes of feasible and infeasible instances. The reader is invited to compare these results with the qualifications given in the text.

The list of computer equipment used should help to correct running times for differences in hardware performance. However, for a balanced comparison between the running times needed by the different methods more detailed information about the relative speeds of the machines used would be needed.

3.2.4 Further developments

The literature reports on the design of several local search algorithms for variants of the frequency assignment problems considered in this paper. For example, Castelino et al. (1996) compare implementations of iterative improvement, tabu search and genetic algorithms for a combat net radio link frequency assignment problem, and (Hao et al., 1998)

perform frequency assignment in mobile radio networks by tabu search. In this section we will briefly describe two developments that originated in the CALMA project.

Approximation: a genetic algorithm with exact crossover. Later refinements of the genetic algorithm of UM include an optimization routine for the crossover. This sort of gene engineering method constructs an optimal offspring by solving an integer linear programming formulation of the crossover problem using a cutting plane algorithm (Kolen, 1997). The cutting planes that are added to a linear relaxation of the problem correspond to so-called 3-cycle inequalities. The algorithm terminates when an integer solution is obtained or no further violated inequalities are found. In the latter case the remaining problem is solved by CPLEX, a commercial branch-and-bound code. Essential for the success of this approach is the extensive preprocessing applied to the integer programming formulation so as to eliminate dominated links and redundant constraints and to reduce the domains.

Approximation: a gradient descent method. Warners (1999) describes a gradient descent method for MSUP which outperforms his potential reduction algorithm. The two methods differ in the way a descent direction is determined. Where the latter algorithm uses a computationally involved method based on the second-order derivatives, the gradient descent method confines itself to the use of the gradient for that purpose.

An efficient implementation of the potential reduction algorithm is reported in Pasechnik (1998).

Optimization: a tree decomposition. Koster (1999) recently proposed a *dynamic programming* algorithm for the minimum interference problem. It decomposes the problem into smaller subproblems and extends these gradually to the overall problem. The decomposition is based on the observation that the optimal choice of a frequency for a link in MIP only depends on the frequencies assigned to the links interfering with it. In general, if S is a separating vertex set of the interference graph G and if an optimal assignment for S is known, then the optimal assignments for all connected components separated by S only depend on S and can be computed independently of each other. The problem has to be solved for every feasible assignment for S, because an optimal assignment for S is not known beforehand.

The algorithm starts by computing a *tree decomposition* of G (Robertson and Seymour, 1986). It yields a cover of G by a set of subgraphs corresponding to the nodes in a tree. The vertices of such a subgraph form a vertex separating set. Two nodes in the tree are connected via a path if their subgraphs intersect. Based on this decomposition, the initial subproblems and the order in which they are extended are determined. The initial subproblems are given by the subgraphs corresponding to the leaf nodes in the tree decomposition. Subsequently, the subproblems are extended by adding new nodes to them or by merging two subproblems.

Table 3.1: Computational results for CELAR instances

method	group	minimum spectrum usage							minimum interference					
		1	2	3	4	5	11	time	6	7	8	9	10	time
SA (simulated annealing)	TUE	2	0	0	0	0	2	1	7%	65%	5%	0%	0%	310
TS (tabu search)	TUE	2	0	0	0	–	0	5	–	–	–	–	–	–
VDS (variable-depth search)	TUE	2	0	0	0	0	2	6	4%	0%	14%	0%	0%	85
SA (simulated annealing)	CERT	4	0	0	0	–	10	41	44%	1299%	70%	2%	0%	42
TS (tabu search)	KCL	2	0	0	0	0	2	40	170%	1804%	566%	8%	1%	111
GENET (see Section 2)	KCL	0	0	0	0	0	2	2	14%	27%	40%	–	–	20
GA (genetic algorithm)	UEA	6	0	2	0	i	10	24	1%	386%	134%	3%	0%	120
GA (genetic algorithm)	UM	–	–	–	–	–	–	–	1%	0%	0%	0%	0%	hrs
PCS (partial constraint satisfaction)	CERT	4	0	6	0	0	–	28	86%	2563%	246%	47%	12%	6
PR (potential reduction)	TUD	0	0	2	0	0	–	3+	29%	–	–	4%	1%	10+
BC (branch-and-cut)	TUD,TUE	0*	0*	0*	0*	0*	0*	10+	–	–	–	–	–	–
CS (constraint satisfaction)	UM	0*	0*	0*	0*	0*	0*	hrs	–	–	–	–	–	–
0–1 QP (0–1 quadratic programming)	TUE	–	–	–	–	–	–	–	–	–	–	−5%	−1%	hrs
best solution		16*	14*	14*	46*	792*	22*		3389*	343592	262	15571*	31516*	

Table 3.2: Computational results for surprise instances

method	group	minimum spectrum usage									minimum interference								
		1	2	3a	4a	8	9	10	14	time	3	4	5	6	7	11	12	13	time
SA	TUE	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
TS	TUE	0	0	0	0	0	4	4	0	3	–	–	–	–	–	–	–	–	–
VDS	TUE	–	–	–	–	–	–	–	–	–	0%	0%	1%	2%	0%	14%	0%	10%	63
SA	CERT	4	4	–	–	–	–	–	–	22	28%	–	–	127%	23%	–	–	–	81
TS	KCL	0	2	–	–	4	4	–	2	28	–	–	409%	20%	303%	3034%	186%	1911%	min
GENET	KCL	0	0	–	–	2	4	–	–	5	–	–	–	–	–	–	–	–	–
GA	UEA	2	2	i	i	–	10	i	4	9	156%	346%	33%	288%	39%	884%	29%	387%	20
GA	UM	–	–	–	–	–	–	–	–	–	–	–	0%	5%	0%	–	0%	125%	hrs
PCS	CERT	6	2	0	0	–	12	0	12	6	–	–	3305%	919%	293%	2219%	217%	827%	5
PR	TUD	0	0	–	–	–	12	–	0	5	–	–	49%	–	–	–	–	–	20
BC	TUD,TUE	0*	0*	–	–	–	16	–	16	min	–	–	–	–	–	–	–	–	–
CS	UM	0*	0*	0*	0*	–	0*	0*	0	hrs	–	–	–	–	–	–	–	–	–
0–1 QP	TUE	–	–	–	–	–	–	–	–	–	–	–	–	–	−53%	–	–	–	min
best solution		18*	14*	380*	394*	20	18*	394*	10		39	46	221*	4123*	4324*	3080	11827*	10110	

minimum spectrum usage
- 2 number of frequencies above optimum
- i infeasible solution is reported

minimum interference
- 6% deviation from the best known solution

- – the method is not applicable, or no solution is reported
- * optimality of the solution is proved

average running times
- 41 minutes of computation time
- sec seconds of computation time
- min minutes of computation time
- hrs hours of computation time
- \+ preprocessing time is not included

computer hardware
- CERT SUN SPARC 10
- TUD HP9000/720
- TUE SUN SPARC 4
- KCL DEC Alpha 3000 (130 MHz)
- UM (GA) DEC OSF/1 AXP
- UM (CS) PC 486
- UEA DEC Alpha (133MHz)

All feasible solutions are evaluated for each subproblem. Extensive preprocessing is applied to reduce its size, and bounds are used to curtail its solution space. An upper bound for a subproblem is calculated by subtracting from the best known overall upper bound the values of the lower bounds of the solved subproblems that are vertex disjoint from the given one. Similarly, a lower bound is derived from the lower bounds of the subproblems that are completely contained in the given subproblem.

This algorithm has been successfully applied to the CELAR instances, solving three of them to optimality and obtaining strong lower bounds for the others.

3.2.5 Conclusions

Frequency assignment problems form a relatively new class of practical problems, to which many of the techniques of combinatorial optimization can be applied. The algorithmic work by the six groups involved in the CALMA project leads to the following conclusions.

(1) The instances of the minimum interference problem appear to be harder than the ones of minimum spectrum usage problem.

(2) Standard *local search* has benefits for everyone. On the one hand, straightforward codes that incorporate little structural information give reasonable solutions in moderate running times. On the other hand, more sophisticated implementations that employ problem-dependent information in their neighborhood function and search strategy produce better results, often in less time.

(3) With *genetic algorithms* the distinction becomes more marked. The approach must be tuned to the problem at hand in order to make it work, and then it can work very well.

(4) For methods based on (hyper)neighborhood search, it is worthwhile to investigate *hybrid variants*, in particular those that embody some form of bilevel search.

(5) The *potential reduction* method is a new element in the array of approximation algorithms. It evidently has great potential.

(6) In case of *optimization*, the general and tailored approaches are drawn even farther apart. Complete enumeration of all possible solutions is obviously out of the question. In order to find provably optimal solutions, algorithms must incorporate highly specialized elimination rules, based, e.g., on polyhedral techniques or consistency arguments.

(7) Branch-and-bound and constraint satisfaction are two sides of the same coin. Exploring the combination of concepts and techniques from *mathematical programming* and *artificial intelligence* is a promising topic of investigation.

(8) Overall, there is a strong positive correlation between the amount of problem-specific information used, the extent to which mathematical insight is exploited, the development and implementation effort required, and the quality of the results obtained.

On the basis of a broader experience, we venture to suggest that the validity of our conclusions is not restricted to frequency assignment problems but extends to many difficult combinatorial problems arising in planning and design.

3.3 Local search for RLFAP

Local search is a powerful tool to obtain approximate solutions to hard combinatorial optimization problems (Aarts and Lenstra, 1997). The basic variant is *iterative improvement*: given an initial solution $\mathcal{A}$ in the solution set $\mathcal{S}$ with objective value $z(\mathcal{A})$, search its *neighborhood* $\mathcal{N}(\mathcal{A}) \subset \mathcal{S}$ for a solution of lower value, and repeat until no further improvement is possible; the final solution obtained is optimal within its neighborhood, or *locally optimal*. Neighbors of a solution are usually obtained by performing small transformations to it, called *moves*.

The two main issues in local search are the design of effective neighborhood functions $\mathcal{N}$ and the design of search strategies that are able to escape from poor local optima. We focus on the former aspect: designing neighborhoods that exploit the structure of the RLFAP and implementing them in the frameworks of simulated annealing, tabu search and variable-depth search. We first briefly describe these three strategies.

Simulated annealing (Aarts and Korst, 1989) (see Figure 3.1) modifies a solution to a random one chosen from its neighborhood. Improvements are always accepted. Deteriorations are accepted with a certain probability, which depends on the increment of the objective value and a control parameter T. Every K iterations T is decreased according to a rule that depends on σ, the standard deviation of the cost values of solutions visited in the previous K iterations, and δ, a parameter controlling the rate of decrement of T (Van Laarhoven et al., 1992). The function *random* generates uniformly distributed random numbers in the interval $[0, 1]$. The algorithm terminates when T becomes smaller than a given parameter T_n.

---

simulated annealing ($\mathcal{A} \in \mathcal{S}$)
* input: $\mathcal{A} \in \mathcal{S}$;
output: $\mathcal{B} \in \mathcal{S}$, $\mathcal{B}$ is the best solution found during the search. *\
begin
$T := T_0$; $\mathcal{B} := \mathcal{A}$;
repeat
for $i := 1$ **to** K
randomly generate solution $\mathcal{G} \in \mathcal{N}(\mathcal{A})$;
if $random() < exp\ (-(z(\mathcal{G}) - z(\mathcal{A}))/T)$ **then**
$\mathcal{A} := \mathcal{G}$;
if $z(\mathcal{B}) > z(\mathcal{A})$ **then** $\mathcal{B} := \mathcal{A}$;
$T := T/(1 + (T \ln(1 + \delta)/3\sigma))$;
until $T < T_n$;
end

Figure 3.1: Simulated annealing

Tabu search (Glover and Laguna, 1997) (see Figure 3.2) implements a natural strategy of always moving to the best neighboring solution. In order to avoid cycling, the method tries not to return to solutions it has already visited. This idea is implemented by maintaining a tabu list where recently visited solutions are recorded. In order to avoid storing and comparing complete solutions, attributes of such solutions can be used instead. Most commonly the reversal of recently performed moves are put on the tabu list. This may be too restrictive and lead to a situation in which none of the possible moves is allowed. In order to avoid such a deadlock, a so-called *aspiration criterion* can be used to overrule the tabu status of a move, subject to the condition that it does not lead to a short cycle in the search path. The simplest example of an aspiration criterion is that a move generates the best solution so far in the search. Such a move is safe, because it is guaranteed that the resulting solution has not yet been visited. In the sketch of the algorithm in Figure 3.2 we assume that the aspiration criterion is implemented during the selection of the best non-tabu neighbor.

In addition to the standard implementation of tabu search we also use a backtracking mechanism proposed by Nowicki and Smutnicki (1996). The idea is to restart the search from the best solution found if no improvement was found in a long number of iterations. For this purpose, a list of neighbors of a limited set of best-found solutions is maintained, and each time the search is restarted from a random member of the list.

---

tabu search ($\mathcal{A} \in \mathcal{S}$)
* input: $\mathcal{A} \in \mathcal{S}$;
output: $\mathcal{B} \in \mathcal{S}$, $\mathcal{B}$ is the best solution found during the search. *\
begin
$\mathcal{B} := \mathcal{A}$;
repeat
select best non-tabu neighbor $\mathcal{G} \in \mathcal{N}(\mathcal{A})$;
$\mathcal{A} := \mathcal{G}$;
update tabu list;
if $z(\mathcal{B}) > z(\mathcal{A})$ **then** $\mathcal{B} := \mathcal{A}$;
until *stop*();
end

Figure 3.2: Tabu search

Variable-depth search was introduced as a highly problem-specific exchange algorithm for uniform graph partitioning by Kernighan and Lin (1970) and later for the traveling salesman problem by Lin and Kernighan (1973). In the latter article they gave general guidelines for possible extensions of the algorithm to other combinatorial optimization problems. For recent applications to a vehicle routing problem and to the job shop scheduling problem, see Van der Bruggen et al. (1993) and Balas and Vazacopoulos (1998), respectively.

The method can be summarized as follows. Start with an initial solution and perform a sequence of small modifications. Not all such modifications have to be improving. Small deteriorations early in the sequence can be rewarded by major improvements later on. A rule for generating such a sequence can be based on the repeated application of a properly defined neighborhood function $\mathcal{N}$. It has to be ensured, though, that the moves are not reversed within the same sequence. Improving solutions encountered during one such sequence are registered. Typically the sequence is terminated when no gain is expected from further transformations and when the sequence becomes too long or results in a large deterioration of the objective value. The following iteration starts from the best solution obtained in this way.

We propose two interpretations of this method, one for the minimum interference and one for the minimum spectrum usage problem. In both cases, we elaborate on the idea of extending a solution to the edge coloring problem by modifying the colors assigned to the edges along a certain path, as is done in the proof of Vizing's theorem (Berge, 1976). An iteration of our algorithms starts with changing the frequencies of a set of links in the current solution. The new assignment often violates extra interference constraints. An attempt is then made to repair such violations by modifying the frequencies of links along certain trees in the interference graph (see Section 3.3.1). Details are given in Sections 3.3.2 and 3.3.3. Specific features of our algorithms include a problem-specific way of constructing the sequence of changes, based on our definition of neighborhood function, the design of a gain function, which measures the usefulness of a sequence of changes, and the design of a locking rule, which limits the sequence and prevents it from cycling.

3.3.1 Preprocessing and data structures

In the real-life instances we dealt with we observed that links forming a duplex pair show similar interference patterns with other links and that the constraint involving both frequencies is very restrictive. We therefore decided to use the pairs of duplex links as atomic objects in our representation. That is, we substituted the original frequency domains by domains of pairs

$$D_{(i,j)} = \{(f_i, f_j) \mid f_i \in D_i,\ f_j \in D_j,\ |f_i - f_j| = d_{ij}\},\ \forall (i,j) \in L.$$

We say that two pairs of duplex links *interfere* if there is at least one interference constraint defined between the links in the pairs. Note that the set L in the formulation of RLFAP contains pairs of duplex links. We construct an *interference graph* $G = (L, E)$, with a node for each pair of links and an edge between any two interfering nodes. Every node gets a domain of frequency pairs assigned to it. A solution to RLFAP corresponds to an assignment of frequency pairs to the nodes from their domains, respecting the constraints between adjacent nodes. In our representation of the minimum interference problem the edges are defined for both soft and hard constraints, but we distinguish between the two. The graph is used to maintain *arc consistency* (Tsang, 1993) for RLFAP throughout the search. Here, arc consistency is the property that frequency domains contain elements

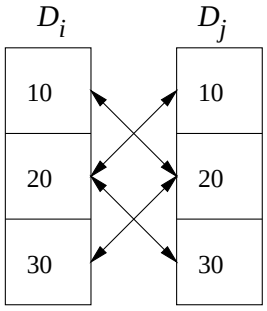


Figure 3.3: Data structure for domains of duplex links ($d_{ij} = 10$).

consistent with the current assignment $\mathcal{A}$. Let us define

$$D^{\mathcal{A}}_{(i,j)} = \{(f_i, f_j) \in D_{(i,j)} \,|\, |f_i - f_{\mathcal{A}}(k)| > d_{ik}, |f_j - f_{\mathcal{A}}(k)| > d_{jk}\ \forall((i,j),(k,.)) \in E\},\ \forall(i,j) \in L,$$

where $f_{\mathcal{A}}(k)$ denotes the frequency assigned to link k in solution $\mathcal{A}$. Now, if we were to modify the frequency pair of any single node (i,j), then we can pick any of the pairs in the domain $D^{\mathcal{A}}_{(i,j)}$ without creating any interference in the current solution.

To implement the arc consistency check efficiently, we store the frequency domains of pairs as two ordered lists of frequencies, one for each link. Every element in such a list contains at most two pointers to its mates, as shown in Figure 3.3. In all real-life instances available to us, each frequency in the domain of a link had exactly one mate frequency, i.e. a frequency satisfying equality constraint 3.1, in the domain of its duplex link. Initially, we assume that all domains are arc consistent. When we change the frequency pair of one of the nodes, we have to enforce arc consistency on all of its adjacent nodes. These is done by creating a copy of the original domain for each of these nodes and removing all of the frequencies inconsistent with the new assignment.

For the minimum interference problem, we also use this data structure to perform local updates of the cost function, once a frequency pair of a node has been reassigned.

3.3.2 The minimum interference problem

An instance of the RLFAP is first interpreted as an instance of the minimum interference problem. If an algorithm for this problem terminates with a solution of zero cost, then we use it to initiate an algorithm for the minimum spectrum usage problem. Otherwise, an approximate solution to the minimum interference problem is reported.

Initial solution

An essential property of the minimum interference problem that we are going to use is its loose constraint structure. That is, it is easy to find a feasible solution, and once it is found there are many ways to transform it into another, almost identical, feasible solution. In the real-life instances of the minimum interference problem that were available to us, all

constraints of type (3.2) are soft. This implies that to each node we can assign a pair of frequencies that respect hard constraints of type (3.1) and guarantee the feasibility of the resulting solution. In case not all interference restrictions are soft and an initial feasible solution is not easy to find, we relax all hard interference restrictions and add a high penalty for their violation to the objective function.

We experimented with various starting procedures for the local search algorithms. We let them begin either directly from a random solution or from a random solution subjected first to an iterative improvement algorithm. Despite our expectations, the latter variant provided better results. We attribute this, however, to the special cost structure of the problem.

Neighborhood

For a given solution $\mathcal{A}$ we define the cost $z_k(\mathcal{A})$ of a node $k \in L$ as its mobility cost plus half the sum of the interference costs for all violated soft interference constraints involving k. Note that the objective value of $\mathcal{A}$ is equal to the sum of the contributions of individual nodes. We define the neighborhood of $\mathcal{A}$ as the set of all solutions that can be obtained by reassigning the frequencies of one node with nonzero cost. We denote the corresponding neighborhood function by $\mathcal{N}$.

A neighborhood function is called *connected* if from every starting solution an optimal solution can be reached using moves defined by this function.

Proposition 1. *The neighborhood function $\mathcal{N}$ is connected.* □

Proof Consider an arbitrary starting solution $\mathcal{A}$, and let $\mathcal{A}_{\text{opt}}$ be an optimal solution. If the objective values of $\mathcal{A}$ and $\mathcal{A}_{\text{opt}}$ are equal, then $\mathcal{A}$ is also optimal. Otherwise, as long as there exists a node k of nonzero cost in $\mathcal{A}$ whose frequencies are different in $\mathcal{A}$ and $\mathcal{A}_{\text{opt}}$, change the frequencies of k in $\mathcal{A}$ to its frequencies in $\mathcal{A}_{\text{opt}}$. Now all the nodes with nonzero cost in $\mathcal{A}$ have the same frequencies in $\mathcal{A}$ and $\mathcal{A}_{\text{opt}}$. By the definition of cost all such nodes have the same cost in $\mathcal{A}$ and $\mathcal{A}_{\text{opt}}$. Hence, the sum of costs over all nodes in $\mathcal{A}$, which is the total cost of $\mathcal{A}$, cannot be larger than the cost of $\mathcal{A}_{\text{opt}}$. Thus starting with an arbitrary solution we construct an optimal solution using only moves to neighboring solutions, which proves the proposition. □

We use this neighborhood function in each of the local search strategies described in the previous section.

Search strategies

Having specified the neighborhood function, we have done the largest part of the job. We will now discuss our attempts to implement tabu search, we will specify the values of the various parameters used in our implementation of simulated annealing, and we will describe how we used the neighborhood function to implement our version of variable-depth search.

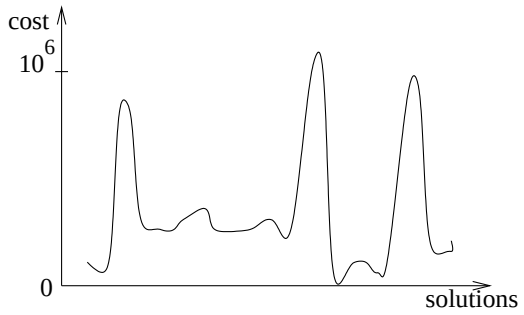


Figure 3.4: Typical cost profile of the minimum interference problem.

Tabu search We tried various settings for the tabu search routine. We experimented with dynamically updated tabu lists and the use of backtracking, but none of these devices enabled us to achieve any satisfactory performance on the test instances.

We attribute the problems we encountered in implementing tabu search for the minimum interference problem to the peculiar cost structure of our test instances, illustrated in Figure 3.4. Typically, these instances contain about 10^{20} of equally good solutions located in between sharp peaks. In our experiments, tabu search always got trapped in one of the lower regions, and even a tabu list of hundreds of elements was not long enough to help it out. After several attempts to scale the cost function of the instances and to adjust the length of the tabu list dynamically, we abandoned the idea of using tabu search for the minimum interference problem altogether. We note that the results reported by Hao et al. (1998) show that a very similar implementation of tabu search may perform well on a different type of instance.

Simulated annealing We used a dynamic cooling schedule. The initial value of the control parameter is determined by a desired initial acceptance ratio. This is achieved by performing a bisection search on T, until a value is found for which, in our case, 90% of the solutions generated is accepted. Furthermore, we used $\delta = 10$ and $K = 2000$ in our implementation of the cooling schedule.

Our simulated annealing algorithm also encountered a difficulty on the CELAR instances of the minimum interference problem. It was able to traverse the solution space freely only at extremely high values of the control parameter. As it became smaller, the chance of crossing a peak became nil. We managed to tackle this problem by a combination of an appropriate scaling of the objective function and a fast cooling schedule. The former enabled us to reduce the height of the peaks, and the latter helped save time descending the slopes.

Variable-depth search In the design of the variable-depth search algorithm (Figure 3.5) we tried to overcome the difficulties we met with the tabu search implementations. Each iteration of the algorithm starts with a random move, usually resulting in a big jump in its objective value, which is followed by a sequence of improving moves.

At the beginning, we select a random node k with a probability proportional to its contribution to the cost of the currently best solution. The so-called *active set* is comprised of the nodes scheduled for reassignment and a *tabu set* holds nodes whose frequencies have already been modified during the current iteration. In a loop, a random node l from the active set is assigned the locally best frequency pair. In case of improvement, the active set is enlarged to include the nodes adjacent to node l. The loop is terminated if the active set becomes empty, or if a randomly generated number from $[0, 1]$ becomes smaller than the exponent of the relative decrement of the objective function. An iterative improvement algorithm is called after each improving iteration. This combination of variable-depth search and iterative improvement is based on the intuitive idea suggested by the difficulties of tabu search and simulating annealing. Namely, variable-depth search builds a tunnel to a new lower region and iterative improvement is used to descend into it. The overall stopping criterion is a bound on the number of iterations without improvement.

---

variable depth search ($\mathcal{A} \in \mathcal{S}$)
* input: $\mathcal{A} \in \mathcal{S}$;
output: $\mathcal{B} \in \mathcal{S}$, $\mathcal{B}$ is the best solution found during the search. *\
begin
$\mathcal{B} := \mathcal{A}$;
do * solution $\mathcal{A}$ is modified in this loop *\
$\mathcal{A} := \mathcal{B}$;
select a random node $k \in L$ with probability $p_k = z_k(\mathcal{A})/z(\mathcal{A})$;
assign a random frequency pair to k;
active set := *adjacent*(k);
tabu set := $\{k\}$;
while *active set* $\neq \emptyset$ **and** *random*() $< \exp((z(\mathcal{B}) - z(\mathcal{A}))/z(\mathcal{B}))$ **do**
remove a uniformly selected random node l from *active set*;
reassign a frequency pair of l to minimize $z_l(\mathcal{A})$;
if the previous step resulted in an improvement **then**
active set := *active set* $\cup$ *adjacent*(l) $\setminus$ *tabu set*;
tabu set := *tabu set* $\cup \{l\}$;
if $z(\mathcal{B}) > z(\mathcal{A})$
$\mathcal{B} := \mathcal{A}$;
iterative improvement ($\mathcal{B}$);
until *stop*();
end

Figure 3.5: Variable-depth search for the minimum interference problem.

3.3.3 The minimum spectrum usage problem

In the minimum spectrum usage problem we assume that a solution of the minimum interference problem with zero objective value is available. We use this solution to initialize

the local search algorithms for this problem.

We experimented with various neighborhoods. In the real-life instances available to us, we observed that the local search algorithms that perform small moves, like the ones used in the minimum interference problem, are not effective because of a lack of correlation between such moves and decrements of the objective function. Indeed, reassigning the frequencies of a node will rarely reduce the number of frequencies used. In this case, one has to invent a secondary criterion to break ties between solutions with the same objective value. In our experiments, we favored solutions with the least minimum number of nodes assigned the same frequency pair. The results were not very encouraging and we decided to opt for a more direct way to change the number of frequencies used, as described below.

Neighborhood

A neighbor of a given solution $\mathcal{A}$ is obtained by selecting a pair of frequencies used in $\mathcal{A}$ and constructing a solution in which these frequencies are not used. We begin our description of this procedure by introducing an initial ordering π of nodes based on the so-called *saturation degree* (Brélaz, 1973):

1. Let S be equal to the set of nodes L.
2. Select the node k with the smallest degree in the subgraph of the interference graph induced by S. In case of ties, select a node $k = (k_1, k_2)$ with the smallest total interference distance $\sum_{(l_1,l_2)\in S\setminus\{k\}} d_{l_1k_1} + d_{l_1k_2} + d_{l_2k_1} + d_{l_2k_2}$ in this subgraph.
3. Remove node k from S.
4. If S is not empty, then return to step 2, otherwise stop.

Our initial ordering π will be the reverse of the order of removal. The ordering π can be loosely interpreted as an ordering of the nodes in decreasing potential difficulty of assigning a frequency to them.

In the minimum spectrum usage problem we allow for partial assignments. Consider a partial assignment $\mathcal{A}$ in which frequency pairs are assigned to a subset $L' \subset L$, so that all constraints defined on L' are satisfied in $\mathcal{A}$. We try to extend this solution by assigning frequencies to the *free* nodes, i.e. the unassigned nodes, in $\mathcal{A}$ in a greedy way. Algorithm **extend** in Figure 3.6 implements this idea. We again use the notation $D^{\mathcal{A}}_{(i,j)}$ for the arc-consistent frequency domain of a node (i,j) defined in Section 3.3.1. This set is computed by removing from $D_{(i,j)}$ the frequency pairs that violate at least one restriction with the nodes already assigned a frequency pair in $\mathcal{A}$.

Consider a node (i,j) assigned frequencies (f_i, f_j) in solution $\mathcal{A}$. We denote by $I^{(i,j)}_{(f_i,f_j)}$ the set of nodes interfering with the frequencies of (i,j) in the current solution:

$$I^{(i,j)}_{(f_i,f_j)} = \{(k,l) \mid ((i,j),(k,l)) \in E, (f_i,f_j) \text{ interfere with } (f_k,f_l)\},$$

where (f_k, f_l) are the frequencies assigned to (k,l) in $\mathcal{A}$. Then the set of *admissible frequencies* $M_{(i,j)}$ contains those alternative frequencies in the domain of (i,j) which either

extend $(\mathcal{A},\ P)$
* input: $\mathcal{A}$ is a partial assignment, P is a set of unassigned nodes;
output: $\mathcal{A}$ is an extended assignment. *\
begin
$P := P \setminus \{(i,j) | D^{\mathcal{A}}_{(i,j)} = \emptyset\}$;
while $P \neq \emptyset$ **do**
select $(i,j) \in P :\ (i,j) = \arg\min_{(k,l)\in P} |D^{\mathcal{A}}_{(k,l)}|$,
break ties by selecting a node highest in the ordering π;
assign to (i,j) a random pair of frequencies from $D^{\mathcal{A}}_{(i,j)}$
choosing a pair already used in $\mathcal{A}$ if that is possible;
update $D^{\mathcal{A}}_{(k,l)},\ \forall (k,l) \in P :\ ((i,j),(k,l)) \in E$;
$P := P \setminus \{(i,j)\}$;
$P := P \setminus \{(k,l)\} | D^{\mathcal{A}}_{(k,l)} = \emptyset\}$;
end

Figure 3.6: Procedure **extend**.

do not interfere with other nodes if assigned to (i,j) in $\mathcal{A}$ or for which this interference may be repaired:

$$M_{(i,j)} = \{(f_i, f_j) \in D^{\mathcal{A}}_{(i,j)} \mid |D^{\mathcal{A}}_{(k,l)}| > 1,\ \forall (k,l) \in I^{(i,j)}_{(f_i,f_j)}, (f_i, f_j) \text{ are used in } \mathcal{A}\}.$$

Procedure **generate neighbor** in Figure 3.7 illustrates our method to generate neighbors for the minimum spectrum usage problem.

Search strategies

As we mentioned before, we allow for partial solutions in our neighborhood. However, experiments have shown that our local search algorithms tend to become inefficient if the number of unassigned nodes becomes too high. Therefore each time during the search we detect that the number of unassigned nodes is above 10% of the total number of nodes, an appropriate algorithm for the minimum interference problem is called to restore feasibility of the solution. Subsequently, the algorithm for the minimum spectrum usage is resumed with whatever solution is returned.

We used the following definition of the cost function in our implementation of the local search algorithms for the minimum spectrum usage problem:

$$z(\mathcal{A}) = \begin{cases} |\cup_{(i,j)\in L} D_{(i,j)}| + N, & \text{if } N > 0 \text{ nodes are unassigned in } \mathcal{A}; \\ |\cup_{(i,j)\in L} \{f_i, f_j\}|, & \text{otherwise.} \end{cases}$$

In the following paragraphs we specify the parameter setting for our implementation of the local search algorithms.

generate neighbor ($\mathcal{A} \in \mathcal{S}$)
* input: $\mathcal{A} \in \mathcal{S}$;
output: $\mathcal{B} \in \mathcal{S}$, $\mathcal{B}$ is a neighbor of $\mathcal{A}$. *\
begin
select randomly a pair of frequencies ϕ already used in $\mathcal{A}$;
compute the set W_ϕ of all nodes assigned the frequency pair ϕ in $\mathcal{A}$;
let P be the set of nodes unassigned in $\mathcal{A}$;
$\mathcal{B} := \mathcal{A}$;
for $\forall (i,j) \in W_\phi$ **do** * solution $\mathcal{B}$ is modified in this loop *\
if $M_{(i,j)} \neq \emptyset$ **then**
assign to (i,j) a pair of frequencies $(f,g) = \arg\min_{(v,w)\in M_{(i,j)}} |I^{(i,j)}_{(v,w)}|$;
$P := P \cup I^{(i,j)}_{(f,g)}$;
else
$P := P \cup \{(i,j)\}$;
$\mathcal{B} :=$ **extend**$(\mathcal{B}, P)$;
end

Figure 3.7: Procedure **generate neighbor** for the minimum spectrum usage problem.

Simulated annealing We used the same settings as for the minimum interference problem with one exception: now $K = 10$.

Tabu search The tabu list of length 7 contains the seed pair of frequencies used to generate neighbors in the **generate neighbor** procedure. If no improvement is found in $N = 200$ iterations, backtracking is performed to a neighbor of the currently best solution. At most 10 backtrackings are performed during the search.

Variable-depth search In the variable-depth search implementation given in Figure 3.8 we elaborated on our approach for the minimum spectrum usage problem. In a loop we first select a seed pair of frequencies. Then, we compute the set of all nodes assigned this frequency. Subsequently, for each of these nodes we try to find an improving sequence of reassignments. For that, a node is selected at each step and it is assigned a pair of frequencies that creates the minimum number of interferences. This step is repeated for all interfering nodes until all interferences are resolved or their reduced frequency domains become empty. The set *tabu set* is used to avoid multiple reassignments of the same nodes within one iteration.

3.4 Lower bounds for MIP

To assess the quality of approximate solutions generated by the local search algorithms we develop a lower bounding procedure for the minimum interference problem. We formulate

variable depth search ($\mathcal{A} \in \mathcal{S}$)
* input: $\mathcal{A} \in \mathcal{S}$;
output: $\mathcal{A} \in \mathcal{S}$, $\mathcal{A}$ is the best solution found during the search. *\
begin
repeat
select randomly a pair of frequencies ϕ already used in $\mathcal{A}$;
compute set W_ϕ of all nodes assigned the frequency pair ϕ in solution $\mathcal{A}$;
let P be a set of nodes unassigned in $\mathcal{A}$;
tabu set $:= \emptyset$;
$\mathcal{B} := \mathcal{A}$;
for $\forall (i,j) \in W_\phi$ **do** * solution $\mathcal{B}$ is modified in this loop *\
$S := \{(i,j)\}$;
while $S \neq \emptyset$ **do**
select randomly $(k,l) \in S$; $S := S \setminus \{(k,l)\}$;
compute $M'_{(k,l)} = \{(f_k, f_l) \in D^{\mathcal{B}}_{(k,l)} \setminus \{\phi\} \mid I^{(k,l)}_{(f_k,f_l)} \cap \mathit{tabu\ set} = \emptyset,$
(f_k, f_l) are used in $\mathcal{A}\}$;
if $M'_{(k,l)} \neq \emptyset$ **then**
assign to (k,l) a pair of frequencies $(f,g) = \arg\min_{(v,w) \in M'_{(k,l)}} |I^{(k,l)}_{(v,w)}|$,
in case of ties select $(f,g) = \arg\max_{(v,w) \in M'_{(k,l)}} \min_{(i,j) \in I^{(k,l)}_{(v,w)}} |D^{\mathcal{B}}_{(i,j)}|$;
$S := S \cup I^{(k,l)}_{(f,g)}$;
tabu set $:=$ *tabu set* $\cup \{(k,l)\}$;
else
$P := P \cup \{(k,l)\}$;
$\mathcal{B} :=$**extend**$(\mathcal{B}, P)$;
if $z(\mathcal{B}) < z(\mathcal{A})$ **then**
$\mathcal{A} := \mathcal{B}$;
until *stop*();
end

Figure 3.8: Variable-depth search for the minimum spectrum usage problem.

a nonlinear relaxation of the minimum interference problem, so that its optimal solution value is a lower bound on the optimal solution value of the minimum interference problem. We describe a preprocessing technique and an enumeration scheme to handle the problem. Our approach works well if two conditions hold:

- A significant number of links have a preassigned frequency and at least some of these frequencies cannot be changed.
- The mobility coefficients M_i are not much smaller than the interference coefficients C_{ij}.

3.4.1 A nonlinear relaxation

Consider the following decision variables:

$$x_i = \begin{cases} 1, & \text{if link } i \text{ is set to its preassigned frequency } p_i; \\ 0, & \text{otherwise.} \end{cases}$$

These variables are fixed to 1 for all links $i \in HM$, whose preassigned frequency cannot be changed. Note that even if a link does not have any preassigned frequency, we can specify one and set its mobility coefficient to zero without loss of generality. Denote by K_{ij} the cost of interference between i and j if both of them are set to their preassigned frequencies:

$$K_{ij} = \begin{cases} C_{ij}, & \text{if } |p_i - p_j| \leq d_{ij} \ ; \\ 0, & \text{otherwise.} \end{cases}$$

We denote by K_{ifj} the cost of interference between i and j if frequency f is assigned to i and j is set to its preassigned frequency:

$$K_{ifj} = \begin{cases} C_{ij}, & \text{if } |f - p_j| \leq d_{ij} \ ; \\ 0, & \text{otherwise.} \end{cases}$$

Let R be the set of all links. Now consider the following problem:

$$\min \sum_{i,j \in R, i<j} K_{ij} x_i x_j + \sum_{i \in R} M_i (1 - x_i) + \sum_{i \in R} (1 - x_i) \Big(\min_{f \in D_i \setminus \{p_i\}} \sum_{j \in R} K_{ifj} x_j \Big), \tag{3.8}$$

where $x_i \in \{0, 1\} \ \ \forall i \in R$.

We will interpret the objective function for some link i. If the decision is made not to use the preassigned frequency p_i for i, in other words if $x_i = 0$, then the second and the third terms in (3.8) occur for i. The second term is simply the mobility cost of i. The third term is the minimal cost of interference that occurs between i and all other links that are set to their preassigned frequencies. Suppose now that the frequency of i is fixed to its preassigned value ($x_i = 1$). The first term gives the cost of interference between i and all other links fixed to their preassigned frequencies. The variable x_i is also considered in the third term, determining the cost of the cheapest alternative frequency for links whose frequencies are different from their preassigned ones.

The only interference cost that is not covered by this objective function concerns the interference between links with frequencies both different from their preassigned ones. Note that all cost coefficients in this problem are nonnegative. It follows that an optimal solution to the relaxation yields a lower bound for the minimum interference problem.

3.4.2 Preprocessing

In order to solve problem (3.8) we first try to reduce its size. For this purpose we will apply some preprocessing, that is, we fix some variables x_i to 1 or to 0 and try to prove

that there is an optimal solution in which they will have these values. Define by F_0 the set of links i for which x_i has been fixed to 0. In the same manner we define F_1 and F_2 for the variables fixed to 1 and the free variables, respectively. Note that $F_0 \cup F_1 \cup F_2 = R$ and $i \in F_2$.

Suppose we want to fix x_i at 1. Consider an arbitrary solution in which $x_i = 0$. If we are able to show that the objective function of this solution cannot increase when x_i is fixed to 1, we are free to do so. A similar consideration holds for fixing x_i at 0.

We will now derive a sufficient condition for fixing variable x_i to 1 for some link i from F_2. Consider an arbitrary solution with $x_i = 0$. If we change x_i to 1 in this solution, then some cost terms may increase and new ones may appear, on the other hand a part of the objective function will decrease. We call the former phenomenon the *cost* of the change and the latter its *saving*. In order to fix a variable, we have to show that the saving of the change is not smaller than its cost.

The saving obtained by changing x_i from 0 to 1 is equal to

$$M_i + \min_{f \in D_i \setminus \{p_i\}} \sum_{j \in F_1 \cup F_2 \setminus \{i\}} K_{ifj} x_j. \tag{3.9}$$

In words, it consists of a mobility cost of i and an interference cost of the cheapest alternative to a preassigned frequency. We cannot evaluate this expression directly, because it contains decision variables. The principle remains valid, however, if we consider a lower bound on the saving and an upper bound on the cost. The cost incurred by changing x_i from 0 to 1 is given by

$$\sum_{j \in F_1 \cup F_2 \setminus \{i\}} K_{ij} x_j + \sum_{j \in F_0 \cup F_2 \setminus \{i\}} (1 - x_j) T_{ij}, \tag{3.10}$$

where T_{ij} is defined as the cost increase related to a link j with $x_j = 0$:

$$T_{ij} = \min_{f \in D_j \setminus \{p_j\}} \Big(\sum_{l \in R \setminus \{i\}} K_{jfl} x_l + K_{jfi} \Big) - \min_{f \in D_j \setminus \{p_j\}} \Big(\sum_{l \in R \setminus \{i\}} K_{jfl} x_l \Big).$$

Since

$$T_{ij} \leq \min_{f \in D_j \setminus \{p_j\}} \Big(\sum_{l \in F_2 \cup F_1} K_{jfl} \Big) - \min_{f \in D_j \setminus \{p_j\}} \Big(\sum_{l \in F_1} K_{jfl} \Big) =: \delta_j(F_1, F_2) \ ,$$

and

$$T_{ij} \leq \max_{f \in D_j \setminus \{p_j\}} K_{jfi} \ ,$$

we obtain the following upper bound on T_{ij}:

$$T_{ij} \leq \min\{\delta_j(F_1, F_2), \max_{f \in D_j \setminus \{p_j\}} K_{jfi}\} =: \delta_{ij} \ .$$

The value of δ_{ij} can be calculated for every j. The difference between the saving (3.9) and the cost (3.10) is at least

$$M_i + \min_{f \in D_i \setminus \{p_i\}} \sum_{j \in F_1 \cup F_2 \setminus \{i\}} K_{ifj} x_j - \sum_{j \in F_1 \cup F_2 \setminus \{i\}} K_{ij} x_j - \sum_{j \in F_0 \cup F_2 \setminus \{i\}} (1 - x_j) \delta_{ij} =$$

$$M_i + \min_{f \in D_i \setminus \{p_i\}} [\sum_{j \in F_2 \setminus \{i\}} ((K_{ifj} - K_{ij})x_j - (1 - x_j)\delta_{ij}) + \sum_{j \in F_1} (K_{ifj} - K_{ij}) + \sum_{j \in F_0} (-\delta_{ij})] \geq$$

$$M_i + \min_{f \in D_i \setminus \{p_i\}} [\sum_{j \in F_2 \setminus \{i\}} \min\{K_{ifj} - K_{ij}, -\delta_{ij}\} + \sum_{j \in F_1} (K_{ifj} - K_{ij}) + \sum_{j \in F_0} (-\delta_{ij})] \ .$$

Now a sufficient condition for fixing x_i at 1 is that

$$M_i + \min_{f \in D_i \setminus \{p_i\}} \sum_{j \in R \setminus \{i\}} \gamma_{ifj} \ \geq \ 0 \ , \tag{3.11}$$

$$\text{where} \quad \gamma_{ifj} = \begin{cases} -\delta_{ij}, & j \in F_0, \\ K_{ifj} - K_{ij}, & j \in F_1, \\ \min\{K_{ifj} - K_{ij}, -\delta_{ij}\}, & j \in F_2. \end{cases}$$

We repeat the scheme to derive a sufficient condition for fixing x_i to 0. Consider an arbitrary solution with $x_i = 1$. We change the value of x_i in this solution to 0. The cost of such a change and its saving are now given by expressions (3.9) and (3.10), respectively. We now obtain a lower bound on T_{ij}:

$$T_{ij} \ \geq \min_{f \in D_j \setminus \{p_j\}} K_{jfi} \ =: \ \epsilon_{ij}.$$

The difference between the cost (3.9) and the saving (3.10) is at most

$$M_i + \min_{f \in D_i \setminus \{p_i\}} \sum_{j \in F_1 \cup F_2 \setminus \{i\}} K_{ifj} x_j - \sum_{j \in F_1 \cup F_2 \setminus \{i\}} K_{ij} x_j - \sum_{j \in F_0 \cup F_2 \setminus \{i\}} (1 - x_j)\epsilon_{ij} \leq$$

$$M_i + \min_{f \in D_i \setminus \{p_i\}} \sum_{j \in R \setminus \{i\}} \beta_{ifj} \ ,$$

$$\text{where } \beta_{ifj} = \begin{cases} -\epsilon_{ij}, & j \in F_0, \\ K_{ifj} - K_{ij}, & j \in F_1, \\ \max\{K_{ifj} - K_{ij}, -\epsilon_{ij}\}, & j \in F_2. \end{cases}$$

A sufficient condition for fixing x_i at 0 is that

$$M_i + \min_{f \in D_i \setminus \{p_i\}} \sum_{j \in R \setminus \{i\}} \beta_{ifj} \ \leq \ 0 \ .$$

If the sufficient condition for fixing x_i at 1 or 0 is satisfied, we fix this variable, delete index i from the set F_2 and include it in F_1 (or F_0). Note that once F_1 or F_0 is modified, the sufficient conditions for preprocessing of all free variables are strengthened. So, we repeat this preprocessing cyclicly until no free variable can be fixed anymore.

In practice, the specific structure of an instance determines how many variables can be fixed by preprocessing. The favorable conditions formulated in the beginning of this section are satisfied by CELAR instances 9 and 10. For instance 10, we were able to solve the lower bound problem completely by preprocessing, for instance 9 we had to solve relatively small subproblems left after preprocessing by enumeration.

3.4.3 Efficient enumeration

Starting with an initial solution we enumerate all solutions to the problem according to the binary reflected Gray code (Bitner et al., 1976) (Figure 3.9). Successive solutions obtained in this enumeration scheme differ in only one variable. In a Gray code scheme the number of times a variable has to be modified is different for each variable. We order the variables according to the computational work required to update the cost function, and we use this ordering to minimize the total computational work.

x_1	0110011001100110
x_2	0011110000111100
x_3	0000111111110000
x_4	0000000011111111

Figure 3.9: The binary reflected Gray code for $F_2 = \{x_1, x_2, x_3, x_4\}$.

We first describe how the objective function is updated at each step. When the preprocessing stage is completed, we calculate for each link i in F_2 the cost of interference of link i at its preassigned frequency p_i with all links in F_1; denote it by C_{iF_1}. Define $I(i)$ as the set of all links interfering with link i. Suppose all variables have been given a value and the objective function of this solution has been calculated. We also calculate and store some auxiliary values:

$$V_i \;=\; \min_{f \in D_i \setminus \{p_i\}} \sum_{j \in I(i) \cap (F_1 \cup F_2)} K_{ifj} x_j \;, \quad \forall\, i \in F_0 \cup F_2 \text{ for which } x_i = 0.$$

Consider the change of some variable $x_i,\ i \in F_2$, from 1 to 0. The objective function of the solution then increases by the following amount:

$$- \sum_{j \in I(i) \cap F_2} K_{ij} x_j - C_{iF_1} + M_i + \min_{f \in D_i \setminus \{p_i\}} \sum_{j \in I(i) \cap (F_1 \cup F_2)} K_{ifj} x_j$$

$$- \sum_{l \in I(i) \cap (F_0 \cup F_2)} (1 - x_l)[V_l - \min_{f \in D_l \setminus \{p_l\}} \sum_{j \in I(l) \cap (F_1 \cup F_2)} K_{lfj} x_j].$$

The value V_i, which is the fourth term in this expression, is stored. Once x_i has been set to 0, the values V_j have to be recalculated for all $j \in I(i) \cap (F_2 \cup F_0)$ for which $x_j = 0$.

Now consider the opposite change of some variable $x_i,\ i \in F_2$, from 0 to 1. The objective function then increases by

$$\sum_{j \in I(i) \cap F_2} K_{ij} x_j + C_{iF_1} - M_i - V_i + \sum_{l \in I(i) \cap (F_2 \cup F_0)} (1 - x_l)[\min_{f \in D_l \setminus \{p_l\}} (\sum_{j \in I(l) \cap (F_1 \cup F_2)} K_{lfj} x_j + K_{lfi}) - V_l] \;.$$

Here again, after setting x_i to 1, the values V_j have to be recalculated for all $j \in I(i) \cap (F_0 \cup F_2)$ for which $x_j = 0$.

The computational effort for updating the cost function for each link i in F_2 is proportional to

$$|I(i) \cap F_2| + |I(i) \cap (F_1 \cup F_2)|(|D_i| - 1) + \sum_{j \in I(i) \cap (F_0 \cup F_2)} |I(j) \cap (F_1 \cup F_2)|(|D_j| - 1) \;.$$

We put the links in F_2 in nondecreasing order of this quantity. By applying the Gray code to this ordering, as illustrated in Figure 3.9, we then enumerate all solutions in an efficient way.

3.4.4 Branch and bound

We implemented a branch and bound algorithm using the lower bounds derived for the minimum interference problem. The best solution obtained by the approximation algorithms for the minimum interference problem is used as an upper bound. Branching is performed by splitting the domain of a link into two or more parts. If for some node in the first level of the tree the lower bound value is strictly higher than the value of an upper bound, we conclude that the corresponding subset of its domain is not used in any optimal solution. Using this argument for two of the CELAR instances we were able to reduce the domains of the links by approximately 20%.

3.5 Computational results

We implemented our algorithms in C++. The programs were run on a SUN SPARC4 workstation. Our results are presented in Tables 3.3 and 3.4. We used two classes of test instances. One is a real-life data set derived from a practical application at CELAR. These instances were available to all of the CALMA participants from the beginning of the project. A second set contains random instances, which were designed so as to capture the complexity of the real-life instances. All of the instances are available via anonymous ftp to `ftp.win.tue.nl` in the directory `/pub/techreports/CALMA/Instances`.

For the minimum spectrum usage problem, tabu search is slightly ahead of variable-depth search and simulated annealing. For the minimum interference problem, no results are given for tabu search, as has been explained in Section 3.3.2; the honors are equally divided between variable-depth search and simulated annealing. Overall, our variable-depth search implementation appears to be the fastest procedure.

The lower bounds for the minimum spectrum usage problem were found by constraint satisfaction and linear programming based techniques; see Aardal et al. (1996) for an overview. The lower bounds for the minimum interference problem were obtained by Koster and Van Hoesel (1998) using a dynamic programming algorithm. Our lower bounding procedure was only applied to the CELAR instances 9 and 10, providing bounds of 14969 and 32144 respectively; the other instances did not satisfy our assumptions.

3.6 Conclusions

In this paper we presented several local search algorithms for the radio link frequency assignment problem. We focused on the development of problem-specific neighborhood functions, which determine the tactics of the local search algorithms. Various local search

Table 3.3: Computational results for the CELAR instances

inst.	size		upper bound						lower bounds
	var.	constr.	tabu search		variable-depth		sim. annealing		
			value	*time*	value	*time*	value	*time*	
1*	916	5548	16	*705*	16	*1919*	16	*396*	14
2*	200	1235	14	*7*	14	*1*	14	*2*	14
3*	400	2760	14	*60*	14	*48*	14	*72*	14
4*	680	3967	46	*166*	46	*31*	46	*80*	46
5*	400	2598			792	*137*	792	*49*	792
6	200	1322			3532	*456*	3671	*65327*	3389
7	400	2865			344103	*8363*	567949	*26569*	202720
8	916	5744			299	*16534*	276	*716*	194
9	680	4103			15667	*21*	15665	*372*	15665
10	680	4103			32456	*210*	32456	*24*	32456
11*	680	4103	22	*735*	24	*10*	24	*223*	22

Table 3.4: Computational results for the random instances

inst.	size		upper bound						best known
	var.	constr.	tabu search		variable-depth		sim. annealing		
			value	*time*	value	*time*	value	*time*	
1*	200	1171	14	*27*	14	*22*	14	*36*	14
2*	200	1143	46	*49*	46	*2*	46	*32*	46
3*	200	1160	18	*289*	18	*6*	18	*71*	18
4*	200	1143	20	*101*	20	*80*	20	*57*	20
5	200	1125			5	*210*	6	*146*	5
6*	916	5177	38	*260*	42	*286*	40	*321*	38
7*	916	5173	48	*11016*	48	*9035*	48	*10943*	48
8*	916	5262	40	*130*	40	*314*	40	*463*	40
9	916	5183			32	*6380*	13	*10765*	13
10	916	5123			456	*6915*	407	*11435*	407

inst.	instance number
size var.	number of links
size constr.	number of constraints
time	seconds on SUN SPARC 4
*	instance of the minimum spectrum usage problem

strategies, such as tabu search, simulated annealing and variable-depth search, were implemented for each neighborhood function. Their performance was compared on real-life and random test instances. Our local search algorithms perform well in competition with other techniques applied to the RLFAP within the CALMA project (Aardal et al., 1996).

For a given neighborhood, the choice of the best local search strategy depends on the cost structure of the problem and characteristics of the neighborhood such as its size and the complexity of its evaluation.

Acknowledgments

An extended abstract of Section 3.2 has been published as Aardal et al. (1996). An article based on Sections 3.3–3.6 has been accepted for publication in *Telecommunication Systems* (Tiourine et al., 1999). Our research was supported by the EUCLID program, CEPA 6 (Artificial Intelligence), RTP 6.4 (Combinatorial Algorithms for Military Applications) (Hajema et al., 1993), and by the National Science Foundation through the Center for

Research on Parallel Computation, Rice University, under Cooperative Agreement CCR-9120008. We are grateful to the Centre d'Electronique de l'Armement and the Technische Universiteit Delft for providing test instances, to all participants in the CALMA project for their cooperation and for providing information about their methods and results, and to Michel Minoux for his support and involvement throughout the project.

Chapter 4

Case study: Statistical disclosure control

4.1 Introduction

Statistical disclosure control in microdata is a relatively new problem for statistical offices. The problem arises from contradictory objectives with respect to the public release of microdata files. A microdata file consists of records with information collected from individual respondents, typically by means of a survey. With the release of an anonymized version of such a file, the statistical office aims at providing the most detailed information under the condition that no sensitive information from this file can be attributed with certainty to a particular respondent. Clearly, the dilemma is releasing very detailed information at a high risk of misuse of this information versus providing much less detailed information and guaranteeing the privacy of a respondent. For the definitions used and the background of the problem we refer to the work of Willenborg and De Waal (1996).

Disclosure of information in the microdata file occurs when sensitive information in the file is identified with a certain respondent. Therefore, there are two conditions for a disclosure to occur. First, the file must contain sensitive information. Second, it should be possible to identify this information with a respondent. If one of these conditions is not satisfied, we assume that the file is safe from disclosure.

It is up to a statistical office to judge whether the microdata contains sensitive information. We assume that such information is indeed present and that the issue is to prevent its identification with the individuals from a population. In the absence of directly identifying information like a name or an address, a record in the microdata can be identified by a combination of identifying variables, called a key. There are several scenarios described in the literature as to how this identification can occur. The easiest one concerns the population uniques. If someone is unique in a population on a certain key and the corresponding record is present in a microdata file, the respondent can be identified. A more subtle situation occurs when a group of people identical on a certain key also have similar scores of a sensitive variable. In this case, the sensitive information about a member of this group

can be disclosed although no personal identification occurs.

In practice, it is assumed that, if a record in the microdata file has a rare score for a low dimensional key, it is potentially unsafe. This is a natural extension of the concept of uniqueness. Indeed, if someone is unique in the population on a key, then so is the corresponding record in the microdata file, if present. On the other hand, if a group of individuals can be identified on a low dimensional key, then it is likely that either the members of the group are unique on the higher dimensional key or that the sensitive information shared by the group is revealed.

Frequency tables are used to compute the set of the rare combinations. More precisely a frequency table is set up for each potentially unsafe key. A cell in the table gives the number of records that contain the corresponding combination of values of a key. If this number is between one and a certain threshold, defined for each table, then the combination is declared to be unsafe. These unsafe combinations have to be screened before the microdata file can be released for public use. We will consider two protective measures for microdata: *local suppressions* and *global recodings*, as suggested by Willenborg and De Waal (1996). Global recoding is an operation defined for all records in the microdata file. If, for example, each record in the file contains a field specifying the municipality of a respondent, then a possible global recoding will be to replace the value of that field by the corresponding province for all records. Local suppression on the contrary is applied to a single record by replacing the values of some of its fields by "missing." The protective effect of these techniques is determined from the updated frequency tables.

There is no consensus in the community of statistical offices as to what the measure of information loss should be (Willenborg, 1997). De Waal and Willenborg (1995) proposed to use an entropy function, but the exact implementation of this function has yet to be worked out. We model the information loss by a linear function. Under the assumption of independence, most of the information loss functions known from the literature can be represented in this way.

In the following sections we proceed by formulating the microdata protection problem as an optimization problem. We will have to deal with the size issue, in particular, as the microdata sets themselves may be of enormous magnitude.

4.2 Models for microdata protection

We base our modeling approach on the concepts defined by De Waal and Willenborg (1995). Consider a microdata file as a collection R of records r. In the microdata protection problem we are only interested in the part of the file containing identifying information. We denote the index set of identifying fields in a record by F. Let U be a set of unsafe combinations of identifying fields in the microdata. An unsafe combination $u \in U$ has the form $u = (r, S)$, with $r \in R$ and $S \subseteq F$. As we discussed in the introduction, the set U of unsafe combinations is computed from the microdata file using the frequency tables.

We distinguish between two types of critical unsafe combinations in our models: a minimal unsafe combination (minuc) and a maximal unsafe combination (manuc), which

are inclusion-wise minimal, respectively maximal, unsafe combinations. An unsafe combination $u = (r, S)$ is minimal if there is no $T \subseteq F$ with $T \subset S$ (we use the sign $\subset$ for strict inclusion) and $(r, T) \in U$. It is maximal if there is no $T \subseteq F$ with $S \subset T$ and $(r, T) \in U$. A minuc is protected by suppressing any of its entries and therefore is useful in the definition of a model for local suppression. Manucs are used to formulate the global recoding problem.

We have to protect the unsafe combinations from U using global recodings and local suppressions and to do so with minimal information loss. We will give the examples of the application of local suppressions and global recodings in Sections 4.2.1 and 4.2.2. We will proceed by building our model gradually, starting with simple special cases.

4.2.1 A pure suppression problem

We begin by formulating the exact local suppression problem, following De Waal and Willenborg (1998). For each record r with at least one unsafe combination we introduce variables

$$x_{rf} = \begin{cases} 1 & \text{if the content of field } f \text{ in record } r \text{ is replaced by "missing,"} \\ 0 & \text{otherwise ,} \end{cases}$$

where $f \in F$. By setting at least one value of an unsafe combination at "missing" the combination becomes untraceable and therefore is considered protected. Hence, the local suppression problem for microdata file is

$$\begin{array}{lrcll} \min & \sum_{r \in R, f \in F} c_{rf} x_{rf} & & & \\ \text{s.t.} & \sum_{f \in S} x_{rf} & \geq & 1, & \forall (r, S) \in U, \\ & x_{rf} & \in & \{0, 1\}, & \forall r \in R,\ \forall f \in F. \end{array} \tag{4.1}$$

Under some conditions, we may restrict the constraints to minucs only. This is because if the set U is complete with respect to minucs, or in other words if for each $u \in U$ the set U contains all minimal subsets of u, then the set U is protected by local suppressions if and only if the subset of minucs in U is protected. To prove necessity, suppose that all minucs in U are protected by local suppressions. Let $u = (r, S) \in U$ be any unsafe combination not protected by local suppressions. Consider another combination defined by the not suppressed fields of u: $u' = (r, S') : S' \subset S$, where S' is a subset of not suppressed fields of S. By construction, u' is also an unsafe combination, not necessarily in U, not protected by local suppressions. It is also clear that u' must contain at least one minuc from U, which contradicts the fact that none of the fields of u' are suppressed and therefore the minuc is not protected.

Note that there are variations of the local suppression problem that may connect the problems for the records. For instance, one might want to bound the total number of suppressions of field f from above, by β_f, say. Then the additional restriction $\sum_r x_f^r \leq \beta_f$ turns the overall suppression problem into one very big problem.

Example Consider a collection of records and the corresponding list of minucs given in Table 1.

Table 4.1: Collection of records, with minimal unsafe combinations

record	field 1	field 2
1	10	100
2	11	101
3	19	100
4	19	100
5	10	109
6	10	109

unsafe comb.	in record	minuc	protected by suppression
1	1	10×100	x_{11}, x_{12}
2	2	11	x_{21}
3	2	101	x_{22}

An unsafe combination in this example is any unique combination of record field values. The local suppression problem for this data is

$$\begin{array}{lrl} \min & c_{11}x_{11}+c_{12}x_{12}+c_{21}x_{21}+c_{22}x_{22} & \\ \text{s.t.} & x_{11}+x_{12} \geq & 1, \\ & x_{21} \geq & 1, \\ & x_{22} \geq & 1, \\ & x_{11}, x_{12}, x_{21}, x_{22} \in \{0,1\}. & \end{array}$$

4.2.2 A restricted combination of local suppression and recoding

In the formulation below we require that every unsafe combination is protected by either local suppression or global recoding. This can be done, because, in principle, for each unsafe combination we can list all local suppressions and global recodings that protect it.

We introduce the following variables:

$$y_{fk} = \begin{cases} 1 & \text{if the content of field } f \text{ is recoded according to rule } k \\ & \text{for every record,} \\ 0 & \text{otherwise,} \end{cases}$$

for $k \in K_f$, where K_f is the set of possible recodings of field f.

A constraint matrix B with columns corresponding to variables y_{fk} and rows corresponding to unsafe combinations u, characterizes the protection of unsafe combinations by global recodings. The column corresponding to a variable y_{fk} has an entry 1 in row $u \in U$ if and only if unsafe cell u is protected by the global recoding k of field f. To construct this column, we effectuate the corresponding global recoding. Consider an unsafe combination $u = (r, S)$. If, in the recoded microdata set, there are enough records r' with the same score in the fields S, the combination u is considered to be protected. Note that we only apply recoding of one specific field f, and leave all other fields unchanged.

Matrix B constructed in this way characterizes protection effect of the global recodings. This characterization is complete under the assumption that the combinations of global recodings have no added effect. However, this is not always the case. Often a combination of two or more global recodings protects unsafe combinations that are not protected by either of the global recodings. We will illustrate this issue in the example below.

We define a constraint matrix A as the incidence matrix of record fields and unsafe combinations. That is, the coefficient in row u and column f of A is 1 if combination u contains field f, and 0 otherwise.

Now our first step is to consider local suppressions in a restricted combination with global recodings. In this approximation we neglect the possible effects of superpositions of the global recodings. We will clarify this issue below. Then the restricted local suppression and global recoding problem is

$$\begin{array}{lrcll} \min & c_x x + c_y y & & & \\ \text{s.t.} & Ax + By & \geq & \mathbf{1}, & \\ & \sum_{k \in K_f} y_{fk} & = & 1 & \forall f \in F, \\ & x_{rf} & \in & \{0,1\}, & \forall r \in R,\ \forall f \in F, \\ & y_{fk} & \in & \{0,1\}, & \forall k \in K_f,\ \forall f \in F, \end{array} \tag{4.2}$$

where c_x and c_y are the cost vectors corresponding to local suppressions and global recodings, respectively. The equality constraints ensure that every field is recoded in one way. By convention we include the recoding "leaving as it is" as one possibility.

Example (continued). We assume that the fields in the microdata file can be recoded as shown in the following table. The first column of each table gives the original values of the corresponding fields. The second one gives the recoded values. For example, variable y_{12} corresponds to the replacement of the values of Field 1 from $\{10, ..., 19\}$ by the range 10–19. The corresponding local suppression and elementary global recoding problem is

Table 4.2: Recodings of Fields 1 and 2

Field 1

orig. value	recoded value	corresp. variable
10,11	10–11	y_{11}
18,19	18–19	y_{11}
10..19	10–19	y_{12}
10..19	10..19	y_{13}

Field 2

orig. value	recoded value	corresp. variable
100,101	100–101	y_{21}
108,109	108–109	y_{21}
100..109	100–109	y_{22}
100..109	100..109	y_{23}

given by

$$\begin{array}{lllllllllll}
\min & c^x_{11}x_{11}+c^x_{21}x_{21}+c^x_{12}x_{12}+c^x_{22}x_{22}+c^y_{11}y_{11}+c^y_{12}y_{12}+c^y_{21}y_{21}+c^y_{22}y_{22} & & \\
\text{s.t.} & x_{11}+ \quad x_{21}+ \qquad\qquad\qquad\qquad\qquad y_{12}+ \qquad\qquad y_{22} & \geq & 1, \\
& \qquad\qquad x_{12}+ \qquad\qquad y_{11}+ \quad y_{12} & \geq & 1, \\
& \qquad\qquad\qquad x_{22}+ \qquad\qquad\qquad y_{21}+ \quad y_{22} & \geq & 1, \\
& y_{13}+ \qquad\qquad\qquad\qquad y_{11}+ \quad y_{12} & = & 1, \\
& y_{23}+ \qquad\qquad\qquad\qquad\qquad\qquad y_{21}+ \quad y_{22} & = & 1, \\
& x_{11}, x_{21}, x_{12}, x_{22}, y_{11}, y_{12}, y_{13}, y_{21}, y_{22}, y_{23} \in \{0,1\}. & &
\end{array}$$

This model, however, neglects the fact that some manucs which are not secured by elementary recodings may be protected by a combination of them. As an illustration consider unsafe combination 1 of our example, which is protected by the combination of recodings y_{11} and y_{21}, but not by either of them.

4.2.3 A complete formulation

We now formulate a model that is complete in the sense that it combines the suppression and recoding problem, and also takes the combined effects of recodings into account. Let $\mathcal{K} = \{(k_1, k_2, \ldots, k_{|F|}) \mid k_f \in K_f\}$ denote the set of all possible recodings. Then a vector $\underline{k} \in \mathcal{K}$ denotes a particular recoding of the entire microdata set.

We described in the introduction how the unsafe combinations are determined using frequency tables. It is easy to see that for each unsafe combination there is a unique cell in a frequency table which corresponds to it and that there is a unique set of unsafe cells C corresponding to the set U of unsafe combinations. Note that more than one unsafe combination can be mapped to one cell $\alpha \in C$. We therefore define an unsafe cell as a set of unsafe combinations mapped to it.

In addition to 0-1 variables x_{rf} and y_{fk}, we introduce 0-1 variables

$$z_\alpha = \begin{cases} 0 & \text{if unsafe cell } \alpha \text{ is protected by global recodings,} \\ 1 & \text{otherwise,} \end{cases}$$

for $\alpha \in C$. We introduce 0-1 parameters $P(\alpha, \underline{k})$, where $P(\alpha, \underline{k}) = 1$ if the unsafe cell α is protected by a recoding $k = (k_1, k_2, \ldots, k_{|F|})$, and $P(\alpha, \underline{k}) = 0$ otherwise. The problem is now

$$\begin{array}{lll}
\min & \sum_{r\in R}\sum_{f\in F} c_{rf}x_{rf} + \sum_{f\in F}\sum_{k\in K_f} d_{fk}y_{fk} & \\
\text{s.t.} & z_\alpha + \sum_{\underline{k}\in\mathcal{K}} P(\alpha,\underline{k}) \prod_f y_{fk_f} \geq 1, & \forall \alpha \in C, \\
& \sum_{f\in S} x_{rf} \geq z_\alpha, & \forall (r,S) \in \alpha, \quad \forall \alpha \in C, \\
& \sum_{k\in K_f} y_{fk} = 1, & \forall f \in F, \\
& x_{rf}, y_{fk} \in \{0,1\}, & \forall r \in R, \ \forall f \in F, \ \forall k \in K_f.
\end{array} \tag{4.3}$$

4.2.4 Discussion of the model

In principle, the coefficients $P(\alpha, \underline{k})$ of the model are known. The same is true for the cost coefficients c_{rf} and d_{fk}. A major problem is that it is rather hard to define cost coefficients that suitably describe the overall information loss. Especially difficult in this respect is to find a trade-off between the local suppressions and the global recodings.

Another point is that the model does not seem to possess any structure that suggests an efficient solution technique. In particular, the product form of the recoding variables seems hard to work with.

We will use these points to our advantage. We decompose the problem into two, one for the global recoding and one for the local suppression. This is based on the following argument. Each cell in Model (4.3) has to be protected by either recoding or suppression. For an arbitrary recoding $\underline{k}$ we determine the set $C_{\underline{k}}$ of cells not protected by $\underline{k}$. z is the characteristic vector of this set $C_{\underline{k}} = \{\alpha \in C | z_\alpha = 1\}$. For the cells in $C_{\underline{k}}$ we compute an estimate of the cost of local suppressions. In our approach every cell α gets a weight π_α, which can roughly be interpreted as the cost of local suppressions necessary to protect the cell. Therefore, an overall estimate of the suppression costs amounts to the sum of the weights over the cells in $C_{\underline{k}}$.

We prefer to see the uncertainty in the objective function as an extra degree of freedom. One may think of an iterative setting, where the user may adjust the cost coefficients to reflect his preferences about the measure of information loss.

The structure of the model is also justified by the required interface with the existing software. As one of the results of this study, a solver has been developed and incorporated in a decision support system for statistical disclosure control, called ARGUS (Willenborg and Hundepool, 1998). It implies extra limitations and certainly extra challenges for our approach. In the decision support system the information about a problem instance is available to us via the coefficients c_{rf}, d_{fk} and $P(\alpha, \underline{k})$.

Therefore, we think that the model we have chosen is a good compromise between the phenomenon we are studying, the data available to us, and the solution techniques we have in mind. We proceed by describing the solution techniques for this model.

4.3 Solution approach

4.3.1 The relaxed recoding problem

We first obtain a lower bound on the local suppression problem for each record. Let C_r denote the projection of set C to record r: $C_r = \alpha \in C | \exists(r, .) \in \alpha$. Then the local suppression problem for record r is

$$\begin{array}{lrcll} \min & \sum_{f \in F} c_{rf} x_{rf} & & & \\ \text{s.t.} & \sum_{f \in S} x_{rf} & \geq & z_\alpha, & \forall \alpha \in C_r, \\ & x_{rf} & \in & \{0,1\}, & \forall f \in F. \end{array} \tag{4.4}$$

The dual to the linear programming relaxation of this problem is

$$\begin{array}{lrcll} \max & \sum_{\alpha \in C_r} \pi_{\alpha r} z_\alpha & & & \\ \text{s.t.} & \sum_{\alpha \in C_r : (r,S) \in \alpha,\ f \in S} \pi_{\alpha r} & \leq & c_{rf}, & \forall f \in F, \\ & \pi_{\alpha r} & \geq & 0, & \forall \alpha \in C_r. \end{array} \tag{4.5}$$

According to the duality theory of the linear programming, any feasible solution $\pi_{\alpha r}$ to (4.5) provides a lower bound on the value of (4.4).

In the following formulation, we obtain a lower bound on the complete formulation. Let $\pi_\alpha = \sum_{r \in R} \pi_{\alpha r}$. We define the *relaxed recoding problem* as follows:

$$\begin{array}{ll} \min & \sum_{\alpha \in C} z_\alpha \pi_\alpha + \sum_{f \in F} \sum_{k \in K_f} d_{fk} y_{fk} \\ \text{s.t.} & \\ & z_\alpha + \sum_{\underline{k} \in \mathcal{K}} P(\alpha, \underline{k}) \prod_{f \in F} y_{fk_f} \geq 1, \quad \forall \alpha \in C, \\ & \sum_{k \in K_f} y_{fk} = 1, \qquad \forall f \in F, \\ & y_{fk} \in \{0,1\}, \qquad \forall f \in F,\ \forall k \in K_f. \end{array} \tag{4.6}$$

4.3.2 Strategy

Our strategy in tackling the original problem is as follows.

- Compute weights $\pi_{\alpha r}$ for all-one vectors z_α (it corresponds to protection of the unsafe cells by only local suppressions).
- Find — by whatever method — a recoding that together with the estimated costs of suppression, yields a good solution to the relaxed recoding problem, and thereby gives a good approximate solution to the overall problem.
- Given the recoding found in the previous step, compute or approximate the real optimal solution to the suppression problem. If the value is close to the computed lower bound, then we have a solution and an estimate of its suboptimality, and we stop.
- If we are not satisfied with the solution at hand, we may recompute the weights $\pi_{\alpha r}$ based on the outcome of the last step. That is, we solve (4.5) with z_α derived from the last solution. We go back and repeat the solution procedure.

We consider two ways of finding good solutions to our relaxed recoding problem. The first is based on Lagrangean relaxation, the second on local search. The methods are described below.

4.3.3 Lagrange relaxation

The following method for solving the relaxed recoding problem is motivated by the size of the problem. The idea is to solve problem (4.6) to optimality, for each frequency table separately. If this leads to a consistent overall solution, we are done. Otherwise, we will try to enforce consistency by imposing Lagrangean type penalties. Note that the

relaxed recoding problem for each table is relatively small and could be solved by complete enumeration.

In the following, let T denote the set of frequency tables used to indicate unsafe combinations, and let T_f denote the set of tables that contain field f as one of their dimensions. For convenience we may view a table as a collection of cells. For table t, let I_t denote the set of its coordinates. We introduce new variables for global recodings for each frequency table:

$$y_{fk}^t = \begin{cases} 1 & \text{if the content of field } f \text{ is recoded according to rule } k \\ & \text{for a table t,} \\ 0 & \text{otherwise,} \end{cases}$$

Our intention is to use these variables to determine the best recoding for each frequency table. In other words, if only one frequency table t was generated, then y_{fk}^t will represent the alternative recodings for the microdata. Clearly, there can be different recoding chosen based on the different frequency tables. Therefore, these variables can take values inconsistent with each other and our goal is to find an iterative scheme to eliminate such inconsistencies.

First, we reformulate the problem in the following way:

$$\begin{array}{ll} \min & \sum_{t\in T}(\sum_{\alpha\in t}\pi_\alpha z_\alpha + \sum_{f\in I_t}\sum_{k\in K_f} d_{fk}y_{fk}^t\frac{1}{|T_f|}) \\ \text{s.t.} & \\ & z_\alpha + \sum_{\underline{k}\in\mathcal{K}} P(\alpha,\underline{k})\prod_{f\in F} y_{fk}^t \geq 1, \qquad \forall t\in T, \forall \alpha\in t, \\ & y_{fk}^t = \frac{1}{|T_f|}\sum_{t':f\in I_{t'}} y_{fk}^{t'}, \qquad \forall t\in T, \forall f\in I_t, \\ & \sum_{k\in K_f} y_{fk}^t = 1, \qquad \forall t\in T, \forall f\in I_t, \\ & y_{fk}^t \in \{0,1\}, \qquad \forall t\in T, \forall f\in I_t, \forall k\in K_f. \end{array} \tag{4.7}$$

The first equation demands that all table recodings y_{fk}^t are consistent with respect to common fields. In other words, a feasible recoding will have a form $y_{fk}^{t'} = y_{fk}^t,\ \forall t,t'\in T,\ \forall f\in F,\ \forall k\in K_f$. We obtain a Lagrangean relaxation of this problem by bringing this equality system into the objective. For arbitrary λ in $(|T|\sum_{f\in F}|K_f|)$–dimensional real space, let $L(\lambda)$ be defined by

$$\begin{array}{ll} \sum\limits_{t\in T} & \min(\sum_{\alpha\in t}\pi_\alpha z_\alpha + \sum_{f\in I_t}\sum_{k\in K_f}(\frac{d_{fk}y_{fk}^t}{|T_f|} + \lambda_{tfk}(y_{fk}^t - \frac{1}{|T_f|}\sum_{t':f\in I_{t'}} y_{fk}^{t'}))) \\ \text{s.t.} & \\ & z_\alpha + \sum_{\underline{k}\in\mathcal{K}} P(\alpha,\underline{k})\prod_{f\in F} y_{fk}^t \geq 1, \qquad \forall t\in T, \forall \alpha\in t, \\ & \sum_{k\in K_f} y_{fk}^t = 1, \qquad \forall t\in T,\ \forall f\in I_t, \\ & y_{fk}^t \in \{0,1\}, \qquad \forall t\in T,\ \forall f\in I_t,\ \forall k\in K_f. \end{array} \tag{4.8}$$

In principle, the minimization problems in this formulation have exactly the same structure as the original problem, but it is hoped that they are small enough to be solved by

complete enumeration. The overall Lagrangean relaxation can be solved using the classical subgradient algorithm, see for example Minoux (1986).

Although this method has not been implemented, it may be of interest. It can always be used as a tool to find lower bounds for the relaxed recoding problem. Moreover there may be possibilities of using it as a means to find approximate solutions.

4.3.4 A local search approach

The method that has been implemented is based on the principle of local search. Local search algorithms are often chosen to tackle large-scale practical combinatorial optimization problems. These are particularly suitable if there are a lot of feasible solutions. The general idea of local search is to start with an initial solution and iteratively perform small transformations of this solution in an attempt to improve it with respect to a given criterion. The *neighborhood* of a given solution is defined as the set of solutions to which a given one can be transformed in a single iteration. Various search strategies are used to continue the search even when no immediate improvement is found in a neighborhood. An exhaustive treatment of these techniques and their applications is given by Aarts and Lenstra (1997).

We define the neighborhood of solution y for problem (4.6) as the set of recodings y' such that

$$\exists j \ \left(\ y'_{jk} = y_{j,k\pm1} = 1 \text{ and } \forall i \neq j \ \forall k \ y'_{ik} = y_{ik} \right).$$

In other words, we move from one solution to another by changing the recoding level of one of the fields to an adjacent one.

We start with a random solution, or a solution constructed in some sensible way. We then modify our solution to a neighboring solution. We recompute the solution value, and if the change proves profitable, we effectuate it, otherwise we try another one. A greedy implementation of this strategy will lead to a so-called iterative improvement algorithm. It will find better and better solutions, until it gets stuck in some local minimum.

We have implemented the following search strategies:

- *Iterative Improvement* starts from a randomly generated recoding, or a recoding given by the user. At each step it modifies the current solution to a neighboring solution of lower cost. The method stops when no better neighbor exists.
- *Repeated Iterative Improvement* restarts the iterative improvement procedure from random recodings.
- *Tabu Search* always moves to the best neighbor. In this way the cost of the solutions generated is not necessarily decreasing. To prevent the method from cycling, the reversal of several recently performed moves is disallowed. A stopping criterion is a maximum number of iterations without improvement.
- *Simulated Annealing* modifies a solution to a randomly generated neighbor. Improvements are always accepted. Deteriorations are accepted with a certain probability, decreasing during the run. The method stops when this probability reaches a certain value.

The right choice of a local search algorithm depends on the relative size of the neighborhoods, the probability that a neighboring solution is better, and the effort it takes to evaluate the new solution. The parameter settings of the various approaches are chosen automatically. The efficiency and quality of the various procedures have to be evaluated by performing experiments and judging the solutions within the context of the expected use of the microdata.

4.4 Results

Statistical disclosure control in microdata gives rise to a constrained decision problem. We work with a mathematical programming formulation of the problem. This approach yields a huge optimization model, the formulation of which requires extensive computations. In this model a set of unsafe combinations has to be protected by application of global recodings and local suppressions at minimum information loss. Each global recoding is characterized by a subset of unsafe combinations protected by it. Local suppressions are used for the unsafe combinations left unprotected by the global recodings.

A practical difficulty of our model lies in the definition of an objective function. Here two different approaches can be used. One is based on the information loss, expressed for example by an entropy function, resulting from global recodings and local suppressions. The other is a subjective assessment by a user as to what the value of a resulting microdata will be for his or her research purposes. These two criteria do not necessarily produce the same result. Moreover, there is no consensus about the exact form of the information loss estimate in the former approach.

This motivated us to construct a cost estimate, where we first calculate the effect of the global recodings and then we estimate the cost of the remaining local suppressions. The estimate is based on the solution to a relaxation of the local suppression problem. In this way we obtain a lower bound on the optimal solution value. We gave an iterative procedure that can be used to strengthen this bound. We proposed a local search approach to obtain a solution to this smaller and computationally less intensive model. Its implementation has been incorporated in a decision support system for statistical disclosure control. This still somewhat cumbersome routine spends about 99% of its run time on cost calculations. Our tests show that the iterative improvement algorithm can be a useful on-line navigation tool for a user to obtain a modified microdata set. Tabu search can perform the same task off-line, and simulated annealing provides a costly alternative.

A Lagrangean relaxation based solution technique as proposed in this article has yet to be implemented and tested.

4.5 Concluding remarks

In this section I evaluate my contribution to the ARGUS project from the point of view of its relevance in assisting a decision maker in the real-life situation as perceived by the

owners of the situation, Willenborg and De Waal (1996). My general opinion is that the approach described above offers a structural view on the situation, but does little in providing a guideline for its improvement. In other words, our approach yields a definition of alternative courses of action, but does not offer enough assistance in navigating among them. I attribute this drawback primarily to the shortcomings at the modeling and implementation phases, and partly to the decisions made at the solution phase.

Formulating a problem in operations research, according to Daellenbach et al. (1983), is equivalent to identifying the four main components of the problem: the decision maker, the objectives, the alternative courses of action, and the environment. In this project the concept of the decision maker was not clearly defined, and as a consequence the objectives were not well defined either. I had to make some rough assumptions about the objectives, as described in Section 4.2, and I planned to evaluate them at a later stage. However, despite my efforts I was not able to perform any serious model validation during the project. In particular, these assumptions I made about the objectives were never checked.

Another shortcoming of the project occurred at the software implementation phase. Our block is designed as a part of the ARGUS system, see Section 4.2.4. The algorithms in our block are fed with the data from the database management block of ARGUS. However, the design and implementation of the interface between the two blocks put serious limitations on the information accessible to our block and on the performance of our algorithms. Serious tests of the algorithms were only possible in an offline mode, when our block was disconnected from ARGUS and fed with random data. This experience emphasizes once again the importance of the structural software development, see Section 2.5.

Under the conditions sketched above, a particular choice of an algorithm does not seem to be so important. Nonetheless, I would suggest to use simulations to get the best out of the situation. This suggestion is based on the following argument. In its final version, the interface with ARGUS provides us with the sole opportunity to evaluate solutions, and it works at such a pace that we are only able to evaluate a handful of them in a reasonable time. This limitation suggests a solution approach which takes the number of solution evaluations it performs into account.

Acknowledgments

This chapter is based on the following publication Hurkens and Tiourine (1998). Our research was supported by the European ESPRIT project 20462-SDC.

Chapter 5

Case study: Industrial cutting

5.1 Problem description

DUMO N.V. is a manufacturer of foam products, which are mainly used in the furniture industry. We studied a production process at a department of DUMO in Goirle, The Netherlands. In this department huge foam blocks are cut into order products of various size and shape. The whole cutting process can be roughly subdivided into two stages. At the first stage, rectangular stock blocks are cut into smaller rectangular blocks. The order products are manufactured from the latter blocks at the second stage. The sizes of the intermediate rectangular blocks are determined by the order products and the technical specifications of machines used in the second stage.

Our attention in this project was focused on the first stage of the production process. The company's production involves over a dozen of different foam types. Cutting each of the foam types is performed independently, and therefore the problem can be decomposed according to foam types. Each foam type requires a cutting plan generated as often as twice per day and such a plan typically includes several dozen order products with demands varying from one to about one hundred. The cutting plans have to be generated online with a maximum delay of about a quarter of an hour. This motivated us to design a compact mathematical model and to develop efficient approximation algorithms for it to be able to compete successfully with the manually constructed cutting plans.

This project is joint work with Cor Hurkens, who designed and implemented algorithms for one- and two-dimensional knapsack problems, see Section 5.3.3.

This chapter is organized as follows. Our mathematical model for the industrial cutting problem at DUMO is given in the following section. This section also covers the matters of model analysis and validation. Section 5.3 describes our algorithmic approach. Sections 5.4 and 5.5 are concerned with implementational issues and results of the project.

5.2 The model

5.2.1 Model formulation

We model the industrial cutting problem at DUMO as a three-dimensional cutting problem in which a set of rectangular order blocks has to be cut from larger stock blocks of different sizes. In this problem we are given

- a set S of stock blocks, each of a specified size $X_k \times Y_k \times Z_k$ and a unit volume cost $c_k,\ k \in S$;
- a set O of N order blocks, each of a specified size $x_i \times y_i \times z_i$, and demand $d_i,\ i \in O$.

Usually, an order block is cut arbitrarily from a stock block. However, sometimes the orientation of an order block in a stock block is essential, e.g. when the density of a stock block is not uniform in all directions. In such situations we introduce an extra restriction on the orientation of an order block in a stock block, we call it an *orientation restriction.*

The cutting is performed using *guillotine cuts*, i.e., straight, uninterrupted cuts parallel to one of the facets of the stock block, which partition a block into two parts. Moreover, we will require that a cut will have a margin of at least μ from the nearest facet parallel to the cut. We assume that all the sizes in the problem are integer numbers not less than μ.

Technical characteristics of machines at DUMO do not allow guillotine cuts to be performed over a length exceeding 2 meters. Typically, Y- and Z-sizes of a stock block k are smaller than 2 meters. If $X_k > 2$m, then k has first to be cut into smaller subblocks by a cut parallel to YZ facet. Guillotine cuts satisfying all aforementioned restrictions are called *feasible guillotine cuts.*

A subblock of a stock block k of size $X_k \times Y_k \times Z_k$ is called an *X-slice* of k if its size is equal to $\chi \times Y_k \times Z_k,\ \chi \leq X_k$. A partition of an X-slice into order and waste blocks, obtained by a repeated application of feasible guillotine cuts, is called a *cutting pattern* for an X-slice. An example of a cutting pattern is given on Figure 5.1. With each cutting pattern j for an X-slice of stock block k we associate a vector of integer numbers

$$a_j = (a_{j1}, a_{j2}, \ldots, a_{jN}, 0, \ldots, 0, x_j, 0, \ldots, 0)^T, \tag{5.1}$$

where $a_{ji} \leq d_i$ is the number of times an order block i is cut and x_j in the $(N+k)$-th position is the X-size of the corresponding X-slice.

Let Ψ_k be the collection of all cutting patterns for X-slices of a stock block $k \in S$, let $\Psi = \bigcup_{k \in S} \Psi_k$, and let integer variables ξ_j denote the number of times a cutting pattern $j \in \Psi_k$ for an X-slice of size $x_j \times Y_k \times Z_k$ is used in a solution.

Now the problem is to construct a cutting plan, where a set of order blocks is cut from a set of stock blocks using feasible guillotine cuts. The goal is to minimize the weighted total volume used, not counting the volume of the rest blocks. We propose the following model for this problem:

$$\min \sum_{k \in S} c_k Y_k Z_k \sum_{j \in \Psi_k} x_j \xi_j \tag{5.2}$$

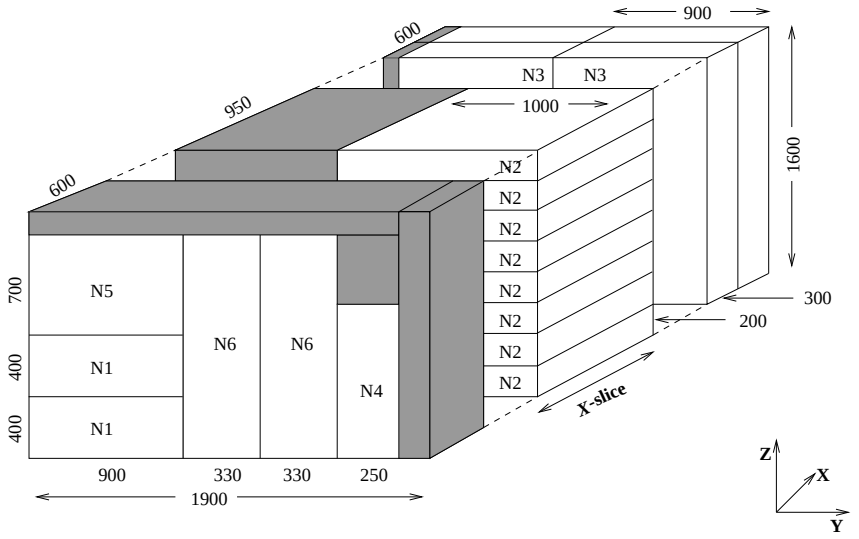


Figure 5.1: Example of a cutting pattern for a stock block of size $2150 \times 1900 \times 1600$.

subject to

$$\sum_{j \in \Psi} a_{ji} \xi_j \geq d_i, \quad \forall i \in O, \tag{5.3}$$

$$\sum_{j \in \Psi_k} x_j \xi_j \leq X_k, \quad \forall k \in S, \tag{5.4}$$

$$\xi_j \geq 0, \text{ integer } \forall j \in \Psi. \tag{5.5}$$

Expression (5.2) gives the weighted total volume of used X-slices. Here, we assume that a rest block in the form of an X-slice can always be reused and therefore it is not wasted. Inequality (5.3) ensures that demand for each item is met. Inequality (5.4) limits the total length of the X-slices from the same stock block.

Models of this type are called *cutting stock* problems. A comprehensive survey of various one- and two-dimensional cutting stock problems and solution techniques for them is given by Haessler and Sweeney (1991). Using the four-field classification of their paper our problem falls in the 3/V/V/R category.

5.2.2 Model validation

In this section we discuss the motivation behind our model and present our arguments for its validity. We developed the model in close cooperation with a project group at DUMO, which included Lieven Pauwelijn, the technical director, Jean-Paul van Beveren, an engineer, and Johan van Boxtel, a senior operator. The general goal of the project was to improve the quality of the cutting process at DUMO. The means included writing a computer program supporting manual as well as fully automatic generation of detailed cutting plans and training the operators to use the program. Such cutting plans have to

be generated with a precision up to millimeters, because of the accuracy of the orders sizes and the technical characteristics of machines.

From a mathematical point of view the model (5.2)–(5.5) is a variant of the so-called *bin-packing problem* (Chvátal, 1983). The bin-packing problem, for which also the term *cutting stock problem* is often used, amounts to cutting orders, in the form of line segments, from identical raws, also line segments, with the least possible waste. A specific feature of our model is that it is three-dimensional. We are not familiar with any published results on the cutting stock problems with three dimensions. Another specific feature is that we discriminate between the raws, or the stock blocks in our case. This is important because in our case the actual stock blocks can have significantly different sizes. Moreover, some of the blocks are more valuable than the others, either because they can be used in other production activities of the company or because they are easier to store. We model this feature by assigning different unit volume costs c_i to the stock blocks. We try to set the unit volume costs in such a way that the use of the less valuable blocks is encouraged but not forced. This also allows us to avoid explicit introduction of binary decision variables describing whether a certain stock block is used in a solution. Indeed, it happens naturally in our model that blocks with smaller c_i are used almost completely, unless doing so is very inefficient.

We paid special attention to model validation in this project. The close cooperation with the project group allowed us to perform the face validity test (see Chapter 1) almost continuously during the modeling phase. For the test of structural validity (see Chapter 1), we list all assumptions and relations incorporated in the model.

1. Only feasible guillotine cuts are allowed.
2. The rest blocks in the form of X-slices can be reused.
3. The input data is precise and known at the time of loading it into the model.
4. The execution of the cutting plans is perfect and without mistakes, so we can neglect the corresponding issue of robustness of our solutions.
5. The demand of the orders has to be met and no backlogging is permitted.
6. The weighted used volume of the stock blocks is a good measure of the efficiency of a cutting plan. Other possible objectives, including the amount of work required to produce a certain cutting plan or the number of stock blocks used, are either neglected, or considered as a constraint in the definition of a cutting pattern, or implied by other restrictions and our goal function.

The first assumption is a direct consequence of the technical specifications of the cutting machines at DUMO. The second one is less objective, but it was unanimously accepted as a realistic assumption by the project group. The third assumption required some investigation. Production at DUMO is performed on order, so all of the orders are known in advance. The problem is that the order sizes are not uniquely defined. They are chosen by engineers of the planning department on the basis of their experience and knowledge of machines and products of the company. However, we reached an agreement with the project group that in the current project we consider the order sizes to be given and fixed,

but we tried to build enough flexibility into our model to allow for variable order sizes to be included in the follow-up project.

Execution of the cutting plans can be considered as perfect up to a good approximation. One argument in favor of this assumption is that machines at DUMO are programmable and each cutting stage (see Section 5.3.3) is performed automatically. Another argument is concerned with solution representation. Although this issue usually is not a part of the modeling phase, it can influence the validity of the fourth assumption and therefore is important in this project. We took this issue into consideration early in the modeling phase, because we thought it is a highly relevant issue in this case and it can be crucial for the success of the whole project. We briefly discuss it here.

In the chosen solution representation (see Section 5.4) a solution is represented by a sequence of cutting stages, each with a clear specification of sizes, cuts, and auxiliary information about particular orders being cut from each subblock. In order to test our solution representation against possible alternatives, we performed experiments on the production floor. The production runs in our experiments were performed by inexperienced operators using solutions suggested by their more experienced colleagues. Various solution representations were tested. Our conclusion was that the chosen representation has the advantage of a natural interpretation as a sequence of tasks performed by an operator. Experiments performed with this representation were in favor of our fourth assumption. The fifth assumption was given to us by the managers.

In order to verify the sixth assumption and to test the predictive power of the model we implemented a prototype of an algorithm for our model. We implemented a kind of truncated branch-and-price algorithm (see Chapter 1) using MINTO, a Mixed INTeger Optimizer (Nemhauser et al., 1994). We performed numerous test runs of this prototype using data from DUMO. Usually, we were given an input data in the evening and we performed our calculations overnight. In the morning, we were rushing to Goirle with our solutions to make the actual production runs. The results of these test runs were very encouraging. First of all, our solutions appeared to be strongly competitive with respect to the manually generated ones. A particular form of these solutions was sometimes contra-intuitive to the experienced operators at DUMO and we took our time to explain the motivation behind such solutions and to convince the operators that they compare favorably with the manually generated solutions. We took several suggestions of the operators into account. These included a restriction on the number of cutting stages in a cutting pattern, which helps to control the amount of work required to produce a cutting plan. The general conclusion of the project group was that the current model satisfactorily reflects the actual production process and that we can proceed with developing a solution method and a decision support system based on this model.

5.3 The algorithm

In our algorithm we construct a partial solution to problem (5.2)–(5.5) and then gradually extend it. For that we alternately solve an LP-relaxation of the problem using a column

generation method and bound one of the variables with a fractional value in the current solution, until an integer solution is found, or no new cutting patterns can be found to extend the current partial solution.

The LP-relaxation is obtained by replacing restriction (5.5) by

$$\xi_j \geq 0, \ \forall j \in \Psi. \tag{5.6}$$

An optimal solution to the LP-relaxation gives a lower bound on the optimal value of the original problem. However, even a relatively small instance of this problem may produce a practically intractable formulation, because of the immense number of cutting patterns in Ψ. Instead, it is possible to obtain an optimal solution to the LP-relaxation for a restricted set of cutting patterns $\Psi' \subset \Psi$ and then add patterns from $\Psi \setminus \Psi'$ to improve upon this solution. Repeated application of this technique will provide an optimal solution to the overall problem. The current solution is optimal if there are no patterns in $\Psi \setminus \Psi'$ to improve upon it.

The problem defined by expressions (5.2)–(5.4) and (5.6) for a set $\Psi' \subset \Psi$ is called a *restricted master problem*. Suppose this problem has a feasible solution ξ^* and let u_i, v_k be the corresponding dual variables associated with constraints (5.3) and (5.4), respectively. Then, the *pricing problem* is defined as

$$\max_{k \in S} \max_{a_j, \ j \in \Psi_k} (v_k - c_k Y_k Z_k) x_j + \sum_{i \in O} u_i a_{ji} \tag{5.7}$$

subject to

a_j corresponds to a cutting pattern $j \in \Psi_k$ for an X-slice of size $x_j \times Y_k \times Z_k, \ x_j \leq X_k$.

From linear programming duality it is known that, if the optimal value of the pricing problem is nonpositive, then ξ^* is an overall optimal solution to the LP-relaxation. On the other hand, any solution to the pricing problem with a positive cost may improve upon ξ^* if the corresponding column (5.1) is added to the restricted master problem formulation; hence the name of this technique: *column generation* (Gilmore and Gomory, 1961).

An outline of our algorithm for problem (5.2)–(5.5) is as follows

1. Initialize the set Ψ' (see Section 5.3.1). Let $\Psi_k^f, \ k \in S$, the sets of variables bounded from above, be empty.
2. Solve the restricted master problem for the set Ψ'. Let ξ^* be an optimal solution.
3. Solve the pricing problem for ξ^*, excluding the variables in $\bigcup_{k \in S} \Psi_k^f$ (see Section 5.3.2).
4. If improving columns are generated in Step 3, add them to Ψ' and return to Step 2.
5. If all non-artificial variables in ξ^* have integer values[1], then stop.
6. Select a stock block $k \in S$
   - such that $\exists \xi_j, \ j \in \Psi' \cap \Psi_k$, with a fractional value in ξ^*,

[1]This means that either ξ^* is a feasible integer solution to the problem (5.2)–(5.5), or there are no columns in Ψ' eligible to extend our partial integer solution.

- with the smallest slack value of constraint (5.4) in ξ^*, and
- with the smallest value $X_k - \sum_{j \in \Psi_k^f} x_j \xi_j^*$.

7. Select a variable ξ_j with a fractional value in ξ^* closest to an integer:

$$\xi_j = \arg \min_{\xi_u,\ u \in \Psi' \cap \Psi_k} \min\{\xi_u^* - \lfloor \xi_u^* \rfloor,\ \lceil \xi_u^* \rceil - \xi_u^*\}.$$

8. If the nearest integer to ξ_j is $\lceil \xi_j \rceil$ and $x_j \lceil \xi_j \rceil + \sum_{u \in \Psi_k^f \setminus \{j\}} x_u \xi_u^* \ \leq \ X_k$, then set the lower bound on ξ_j equal to $\lceil \xi_j \rceil$ and return to Step 2.
9. Set the upper bound on ξ_j equal to $\lfloor \xi_j \rfloor$, add ξ_j to Ψ_k^f and return to Step 2.

In the following sections we will discuss this algorithm in detail.

5.3.1 Initialization

To initialize the column pool Ψ' of the master problem we introduce N artificial variables $\xi_j^a,\ j = 1, \ldots, N$. The columns corresponding to these variables are of the form

$$(0, \ldots, 0, 1, 0, \ldots, 0)^T,$$

where the 1 is in the jth position. The set Ψ' initialized with artificial variables allows us to construct a starting solution. However, such a solution does not correspond to any actual solution to our problem. In order to make artificial variables unattractive in any solution we attach a very high cost to each of them, say $1 + \sum_{i \in S} c_i X_i Y_i Z_i$. Like in the two-phase simplex method (Chvátal, 1983), we expect the artificial variables to have zero values in any feasible solution. However, due to the heuristic nature of our algorithm (see Section 5.2.2), even in case some of the artificial variables have nonzero values in the final solution we cannot conclude that the original problem is infeasible.

5.3.2 Column management

A fundamental difficulty of every column generation algorithm embedded in a branch-and-bound framework is to avoid repeated generation of columns corresponding to the variables in Ψ' on their upper bounds in the current solution (Barnhart et al., 1998). We tackle this problem in a straightforward manner, by simply prohibiting to generate columns for which the corresponding variables in Ψ' are bounded from above. The primary goal of our algorithm is to generate good feasible solutions. Therefore, we also facilitate generation of columns significantly different from those already in Ψ' in order to avoid stagnation of the algorithm.

For each column $j \in \Psi'$ we store the following information:

- the index i of the first nonzero entry a_{ji};
- the sum of all entries $\sum_{i \in O} a_{ji}$;
- x_j, the length of the corresponding X-slice.

Each newly generated column $u \in \Psi_k$ with a positive cost in (5.7) is compared with the columns from Ψ'_k. For this purpose, we define a proximity relation between two columns. A column j is called a *proxy* for column u if they first nonzero entries have the same index and if one of the following three relations holds:

1. $\sum_{i \in O}(a_{ji} - a_{ui}) = 0$, and $x_j < x_u$;
2. $\sum_{i \in O}(a_{ji} - a_{ui}) \geq 0$, and $x_j = x_u$;
3. $\sum_{i \in O} a_{ui} / \sum_{i \in O} a_{ji} = x_u / x_j$.

For example, the first two relations are satisfied in a situation when u is dominated by j, in the sense that either $a_j = a_u$ and $x_j < x_u$, or $a_j \geq a_u$ and $x_j = x_u$. The third relation is satisfied among others by two columns for which

$$\frac{a_{ui}}{a_{ji}} = \frac{x_u}{x_j} \quad \forall i \in O,$$

i.e. the cutting pattern for u is obtained by x_u/x_j repeated applications of the cutting pattern for j. We call such a j a *principal column* for u.

If there are no proxies for u in Ψ'_k, we add u to Ψ'_k. Otherwise, u is rejected unless u itself is a principal column for a column $j \in \Psi'_k$. In the latter case we replace j by u in Ψ'_k.

5.3.3 Approximation algorithm for the pricing problem

For a fixed stock block, the pricing problem is a restricted three-dimensional knapsack problem. In this problem we are given a set of order blocks. For each order block we know its value and demand. The problem is to cut the order blocks from the stock block using feasible guillotine cuts. Each order block may be cut an arbitrary number of times, but not exceeding its demand. The goal is to maximize the total value of the cut order blocks. Needless to say that it is an extremely difficult problem. Moreover, it is a pricing problem in our case and, therefore, it has to be solved very often.

Our approximation approach to this problem is as follows. At the first step, we obtain an approximate solution to the pricing problem for each stock block in S. Usually, a cutting pattern for a stock block can be partitioned into cutting patterns for smaller X-slices. For each stock block we take the finest possible partition. Subsequently, we generate columns corresponding to the cutting patterns of the X-slices in such partitions. If there are columns satisfying the criteria defined in Section 5.3.2, we add several of them to Ψ' at a time.

If no columns are generated in the first step, we solve the pricing problem directly for a number of promising X-slices. For that we select a random order block $j \in O$ with a probability proportional to its value in the current pricing problem and a random stock block $k \in S$. Subsequently, we solve the pricing problem for the following X-slices of the stock block:

$$x_j \times Y_k \times Z_k, \; y_j \times Y_k \times Z_k, \; z_j \times Y_k \times Z_k,$$

or a subset of it satisfying the orientation restrictions (see Section 5.2.1).

Our algorithm for the restricted three-dimensional knapsack problem is a nested family of algorithms in itself. For each instance of the three-dimensional problem we solve at least two instances of its two-dimensional analog (see Section 5.3.3). These are in turn solved by a repeated application of a dynamic programming algorithm for the one-dimensional case (see Section 5.3.3). We describe these algorithms starting from the one-dimensional case.

Restricted one-dimensional knapsack problem with group bounds

This problem is the lowest in our nested family of knapsack problems. We aim at solving this problem to optimality, as fast as possible and with minimum memory requirements.

The model. In the *knapsack problem*, we are given a set of n line segments each of a given length $l_i \in \mathbb{N}$ and a value $p_i \in \mathbb{R}$. Each of the line segments can be cut an arbitrary number of times from a line segment of length $L \in \mathbb{N}$ to maximize the total value. This problem is known to be NP-hard (Garey and Johnson, 1979).

Our experiments with the knapsack problem showed that we have to add extra restrictions to it in order to be able to generate reasonable cutting patterns. Firstly, we have to avoid massive overproduction of items, because otherwise solutions to our LP-relaxation tended to be highly fractional and useless in our efforts to construct good feasible solutions. In an instance of the one-dimensional knapsack problem, there may be several line segments corresponding to various projections of the same three-dimensional order block. For each three-dimensional order block we introduce a group $G_i,\ i = 1, \ldots, N$, and each line segment j in our problem is assigned to exactly one such group with a certain weight w_j. The sum of weights over segments selected from a group G_i should not exceed d_i, the demand of the group. Secondly a cutting machine can only cut segments longer than a certain threshold μ.

Let variables $\zeta_j,\ j = 1, \ldots, n$, denote the number of times a line segment j is cut. The model is given by

$$\Phi(L, n) = \max \sum_{j=1,\ldots,n} p_j \zeta_j \tag{5.8}$$

subject to

$$\sum_{j=1,\ldots,n} l_j \zeta_j + \lambda = L; \tag{5.9}$$

$$\sum_{j \in G_i} w_j \zeta_j \leq d_i,\ i = 1, \ldots, N; \tag{5.10}$$

$$\lambda \in \{0, L\} \cup [\mu, L - \mu];$$

$$\zeta_j \geq 0,\ \text{integer},\ j = 1, \ldots, n.$$

In this problem we want to select the line segments in order to maximize the total revenue given by (5.8). Restriction (5.9) ensures technological feasibility of solutions with respect to the minimum offset μ. The group bounds are observed in (5.10).

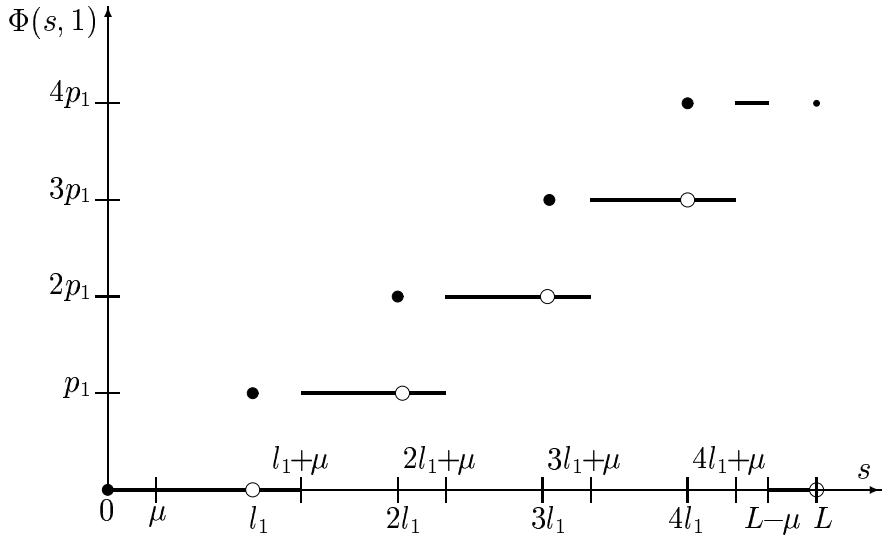


Figure 5.2: Example of the function $\Phi(s,1)$. Fat points denote the *support points* of $\Phi(s,1)$.

Below, we will need to specify instances of the restricted one-dimensional knapsack problem. We will do this by listing the parameters of the problem separated by semicolons in the following order: the groups $G_i,\ i = 1,\ldots,N$, the group bounds $d_i,\ i = 1,\ldots,N$, the length L, the set of items specified by value, length and group weight $(p_j, l_j, w_j), j = 1,\ldots,n$. For example,

$$G_1 = \{1\}, G_2 = \{2\}; d_1 = 4, d_2 = 5; L = 3; \{(p_1 = 1, l_1 = 3, w_1 = 2), (p_2 = 1, l_2 = 1, w_2 = 1)\}.$$

The algorithm. We solve the problem (5.8)–(5.10) to optimality using a dynamic programming algorithm (Martello and Toth, 1990). This algorithm computes optimal solutions for all one-dimensional problems of size $s \le L$, which is essential for the embedding of the algorithm into our algorithm for the two-dimensional case.

For the moment, we assume that there are no group bounds. We begin by ordering the items in nonincreasing unit length value p_j/l_j. Then, we compute the function $\Phi(s_k,1) = kp_1$, for $s_k = kl_1$, $k = 0,\ldots,m_1$, where

$$m_1 = \begin{cases} \lfloor L/l_1 \rfloor - 1 & \text{if } 0 < L \mod l_1 < \mu, \\ \lfloor L/l_1 \rfloor & \text{otherwise.} \end{cases} \tag{5.11}$$

An example of this function is given in Figure 5.2. A pair $(s_k, \Phi(s_k,i))$ is called a *support point* of the function $\Phi(.,i)$ if $s_k = 0$ or if $s_k \ge \mu$ and $\Phi(s_k,i) > \max_{s \le s_k - \mu} \Phi(s,i)$. Knowing the support points, we can compute the function $\Phi(s,i)$ for an arbitrary $s \in \{0, L\} \cup [\mu, L-\mu]$ by

$$\Phi(s,i) \leftarrow \begin{cases} \Phi(s,i) & \text{if } (s,\Phi(s,i)) \text{ is a support point,} \\ \max_{s_k \le s-\mu} \Phi(s_k,i) & \text{otherwise.} \end{cases}$$

We represent the function $\Phi(.,.)$ by its support points.

We proceed by recursively computing the function $\Phi(.,i+1),\ i=1,\dots,n-1$, on the basis of the known support points $(s_k,\Phi(s_k,i)),\ k=1,\dots,m_i$. The following recursive relation is used to compute the support points of the function $\Phi(.,i+1)$:

$$\Phi(s'_k,i+1)=\max\{\Phi(s'_k,i),\ \Phi(s'_k-l_{i+1},i+1)+p_{i+1}\}, \tag{5.12}$$

defined on the ordered list of sizes

$$\{s'_k|\ k=1,\dots,m_{i+1}\}\ =\ \{jl_{i+1}+s_k|\ k=1,\dots,m_i;\ j=0,1,\dots\}\cap[0,L]. \tag{5.13}$$

In the presence of the group bounds things are getting slightly more complicated. We will now have to compute (5.12) for all possible fillings $b=0,\dots,d_j$, for each group $G_j,\ j=1,\dots,N$, in turn:

$$\Phi_G(s'_k,i+1,b)=\max\{\Phi_G(s'_k,i,b),\ \Phi_G(s'_k-l_{i+1},i+1,b-w_{i+1})+p_{i+1}\}, \tag{5.14}$$

where $i,i+1\in G_j$ and s'_k is defined by (5.13). We use this recursive relation in the following way. Items from the same group are treated consecutively in the order of nonincreasing value of p_i/l_i. The function $\Phi_G(s_k,1,b)$ is computed in the following way:

$$\Phi_G(s_k,1,b)=kp_1,\ \text{for } b=0,w_1,\dots,w_1\lfloor d_1/w_1\rfloor,\ s_k=kl_1,\ k=0,\dots,m'_1,$$

where $m'_1=\min\{m_1,\lfloor b/w_1\rfloor\}$ and m_1 is given by (5.11). $\Phi_G(s_k,1,b)=0$ for other values of b. Subsequently, we proceed by computing $\Phi_G(s_k,i,b),\ b=0,\dots,d_1$ for all $i\in G_1$. Unlike in the unbounded case, it is not sufficient to store only the support points while computing this function. We have to remember solutions to all the points defined by (5.13) while treating a certain group. This is because a point which is dominated by a support point may still serve as a good building block for other solutions. This is illustrated with the following example:

$G_1=\{1,2,3\}; d_1=3; L=3;$
$\{(p_1=3,l_1=1,w_1=3),(p_2=1,l_2=2,w_2=1),(p_3=3,l_3=1,w_3=2)\}.$

The optimal solution in this example has value 4 if we take the second and third items once each. But if we start with the first two items in our dynamic programming algorithm, then there will be only one support point of the function $\Phi(s,2)$, namely $(s_1=1,\Phi(s_1,2)=3)$, which corresponds to taking the first item once. Clearly, the point $(s=2,\Phi(2,2)=1)$ will be dominated by the support point. Once we start computing $\Phi(s,3)$, we will notice that there is only a choice between taking the first item once or taking the third item once.

To tackle this problem, we store all intermediate solutions for the points defined by (5.13) while processing a certain group G_j. Once the group has been processed, we compute $\Phi(s'_k,i)=\max_{b=0,\dots,d_j}\Phi_G(s'_k,i,b)$, for all s'_k defined by (5.13), $i\in G_j$. Then, all dominated points can be eliminated, because $\Phi(s_k,i)$ can be represented by its support points. Starting with the first element $i+1$ of the subsequent group G_{j+1} we compute the function $\Phi_G(s'_k,i+1,b),\ b=0,\dots,d_{j+1}, s'_k$ defined by (5.13), using the relation

$$\Phi_G(s'_k,i+1,b)=\max\{\Phi(s'_k,i),\ \Phi_G(s'_k-l_{i+1},i+1,b-w_{i+1})+p_{i+1}\}.$$

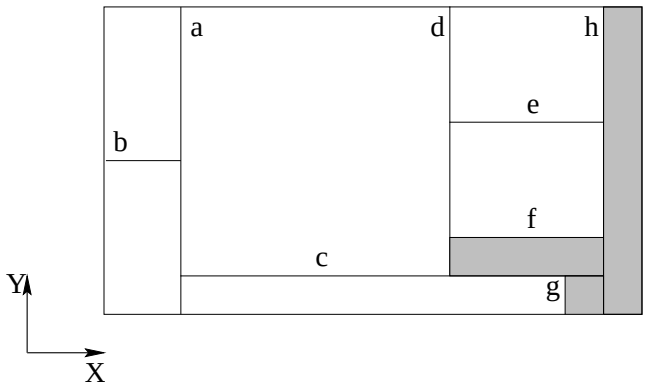


Figure 5.3: Example of a solution with 4-stage cutting.

Restricted two-dimensional knapsack problem with group bounds

This problem is the backbone of our column generation mechanism. Instances of this problem are two-dimensional projections of our original pricing problems, as will be clarified in the next section. The dynamic programming algorithm that we will present, in turn, relies heavily on the algorithm for the one-dimensional case described in the previous section. The aim here is to obtain good feasible solutions, fast and with modest memory use. In the absence of group bounds our algorithm will generate provable optimal solutions if the number of iterations is not restricted. In practice, however, we restrict the number of iterations and treat the group bounds in an approximate way.

The model. The two-dimensional knapsack problem amounts to cutting a given set of rectangular order plates $x_i \times y_i$, $i = 1, \ldots, n$, an arbitrary number of times from a rectangular stock plate of size $X_s \times Y_s$, so as to maximize the sum of the values p_i over the rectangles cut.

As in the one-dimensional case we add several extra restrictions to this formulation. We allow only for the feasible guillotine cuts as described in Section 5.2.1. We will also use group bounds d to limit production of individual items. However, this time our treatment of the group bounds does not allow for such a clean interpretation as in the one-dimensional case.

Furthermore, we omit here a cumbersome mathematical formulation of the problem, and proceed directly with a description of our dynamic programming algorithm.

The algorithm. We start with the definition of a *cutting stage*, a notion essential in our algorithm. Consider an example in Figure 5.3. A cutting pattern in this figure can be represented by the sequence of cuts, for example (a, b, h, c, d, e, f, g). We are interested in a permutation of cuts, with the property that if two cuts c_i and c_j, $i < j$, have the same orientation and can be performed directly after each other, then all cuts $c_k, i < k < j$, in this permutation have the same orientation. A maximal set of cuts

corresponding to an uninterrupted sequence of parallel cuts in this permutation is called a cutting stage. Each solution can be represented as a sequence of cutting stages, in our example $\{a,h\},\{b,c\},\{d,g\},\{e,f\}$. This representation of solutions is unique given an orientation of the first cutting stage. We will use this property here.

Our dynamic programming algorithm builds a solution iteratively, starting from the last cutting stage. At an iteration $\nu > 1$, we compute solutions for all stock plates of sizes $X_i \times Y_i, X_i \leq X_s,\ Y_i \leq Y_s$, limiting the number of cutting stages to ν. In the absence of group bounds we can guarantee optimality of a solution obtained for a stock plate $X_s \times Y_s$ at iteration ν, if solutions for all stock plates $X_i \times Y_i, X_i \leq X_s,\ Y_i \leq Y_s$ are identical at iterations ν and $\nu+1$. We experimented with various settings of our algorithm in the absence of group bounds and concluded that setting a hard limit on the number of iterations to 3 provides stable running times, and yet generates optimal solutions in the majority of our tests.

Algorithms of this type are known from the literature (Gilmore and Gomory, 1965; Hadjiconstantinou and Christofides, 1995). The specific feature of our algorithm is that we do not use a fixed grid in our dynamic programming algorithm, but rely on the support points, introduced in the previous section, in order to store a minimum amount of information. This allows us to handle practical instances with sizes up to 2500×3000, which would be unthinkable to tackle in case of a fixed grid.

Our algorithm works with a fixed orientation of the order plates, i.e., the X- and the Y-sizes of the order plates are not interchangeable.

The initial step of our algorithm corresponds to a cutting stage parallel to the Y-axis. We can choose the orientation of the initial step arbitrarily. This is because this step corresponds to the last cutting stage, which can turn out to be empty in the final solution. This step starts with computing a set $\Theta_1 = \{s_1, \ldots, s_{k_1}\}$ of all different Y-sizes of the order plates. Let $n_j^1,\ j = 1, \ldots, k_1$, denote the multiplicity of each element in Θ_1, i.e., the number of order plates with Y-size equal to $s_j \in \Theta_1$. Subsequently, for each element $s_j \in \Theta_1$ we solve the following instance of the one-dimensional knapsack problem:

$$G; d; L = X_s; \{(p_i, l_i = x_i, w_i)\},\ i \in \{\alpha | y_\alpha = s_j,\ \alpha = 1, \ldots, n\}.$$

For each such problem we record the set of support points $Q_j^1 = \{(\xi, \Phi_j^1(\xi, n_j^1))\}$.

Now we describe an iterative step $\nu > 1$. It starts again with computing a set of interesting sizes:

$$\Theta_\nu = \{s_1, \ldots, s_{k_\nu}\} = \bigcup_{i=1}^{k_{\nu-1}} \{\xi, \xi + \mu |\ (\xi, \Phi_i^{\nu-1}(\xi, n_i^{\nu-1})) \in Q_i^{\nu-1}\}.$$

Θ_ν can be interpreted as the set of all points at which a cut at iteration ν may be of interest. It is possible to show that if there is a cut at $s' \notin \Theta_\nu$, then there is a cut at $s < s',\ s \in \Theta_\nu$ with the same value of the corresponding solution.

The definition of group bounds is not as straightforward as before. At iteration ν, the elementary building blocks are the solutions corresponding to the elements of $\Theta_{\nu-1}$. Such a

building block may contain order plates belonging to different original groups. Therefore, for each composite building block we determine a *critical group* G_i, $i \in \{1, \ldots, N\}$, with the largest ratio of accumulated production within the building block over the group's bound. After that, the whole building block is assigned to a new group G'_i with weight equal to the accumulated production of elements of G_i within the block.

For each size $s_j \in \Theta_\nu$ we solve the following instance of the one-dimensional knapsack problem:

$$\begin{aligned} &G'; d;\ L = \begin{cases} Y_s, & \text{if } \nu \text{ is even,} \\ X_s, & \text{otherwise;} \end{cases} \\ &\{(p_i = \Phi_i^{\nu-1}(s_j, n_i^{\nu-1}), l_i = \vartheta_i, w_i)\},\ \vartheta_i\ \in \Theta_{\nu-1},\ i = 1, \ldots, k_{\nu-1}. \end{aligned} \tag{5.15}$$

Here again we store the support points of the function $\Phi_j^\nu(., n_j^\nu)$ for each subproblem. It is important to notice that we store only the values of solutions corresponding to the support points. The actual solutions can be obtained by backtracking, when necessary. In our implementation we only perform the backtraking to obtain the final solution. Therefore, we have to store additional information about a critical group associated with each support point in order to take the group bounds into account. For each support point, we store the index and a total number of elements of the corresponding critical group.

The principle behind this algorithm can be best understood if one considers an iteration ν as a cutting stage, by cuts parallel to the X-axis if ν is even, and parallel to the Y-axis otherwise. Θ_ν is the collection of all interesting cuts at stage ν. For each value $s_j \in \Theta_\nu$ we solve a subproblem in order to determine the best filling of a slice of size s_j. Our approximation of the optimum value of the original problem with ν cutting stages is given by $\Phi_{k_\nu}^\nu(L, n_{k_\nu}^\nu)$, where L is defined in (5.15). The actual solution corresponding to this value is obtained by backtracking.

Restricted three-dimensional knapsack problem

This problem is the last in the family of the knapsack problems, and it is directly our pricing problem (5.7) for a given stock block. We use an approximation algorithm to solve it.

The model. In the three-dimensional knapsack problem, we are given a set of rectangular order blocks $x_i \times y_i \times z_i$, $i \in O$ with values p_i, and a rectangular stock block $X_s \times Y_s \times Z_s$. The goal is to cut a maximum value subset of order blocks from the stock block.

The extra restrictions are:

- we allow only feasible guillotine cuts, as explained in Section 5.2.1;
- each order block i has its demand d_i, which we treat as a soft upper bound on the multiplicity of the block in a cutting pattern;
- each order block has a restriction on the allowed orientations in the stock block (see Section 5.2.1).

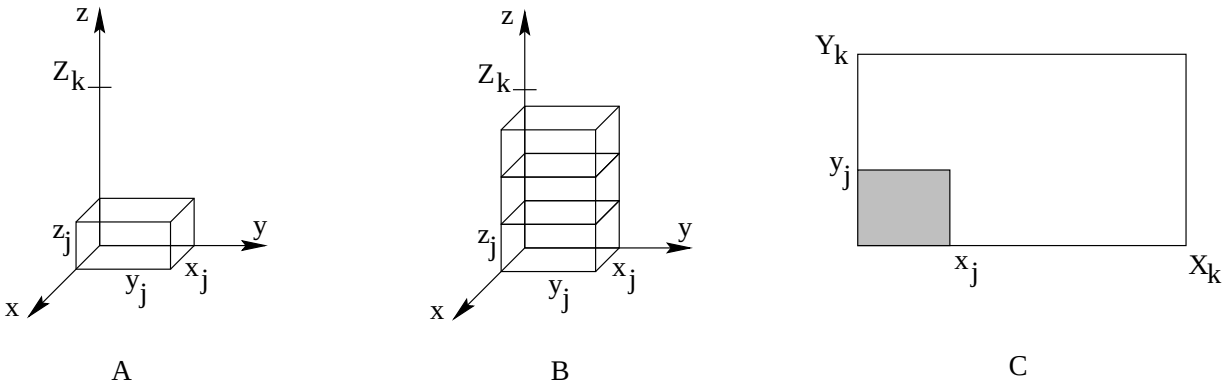


Figure 5.4: Projecting a three-dimensional item on the XY-plane:
A) original item;
B) multi-item in Z direction;
C) XY-projection of the multi-item.

The algorithm. For each instance of the three-dimensional knapsack problem we solve two instances, namely its two-dimensional projections to the XY- and XZ-plane, respectively. At the second stage of our column generation algorithm (see the beginning of Section 5.3.3) we also solve an instance corresponding to the YZ-projection. The orientation of the order plates in our two-dimensional knapsack problem is fixed. In order to overcome this limitation, for each order block we create as many copies as its number of allowed orientations in the stock block, i.e. for an order block of size $x_i \times y_i \times z_i$ we select all allowed permutations of its sizes from the set $\{(x_i \times y_i \times z_i), (x_i \times z_i \times y_i), (y_i \times x_i \times z_i), (z_i \times y_i \times x_i), (z_i \times x_i \times y_i), (y_i \times z_i \times x_i)\}$. All such copies are assigned to the same group G_i. Let the set O' be equal to the set of orders O extended in this way.

Now, consider an arbitrary projection of a three-dimensional instance, say to the XY-plane. An instance of the two-dimensional problem corresponding to this projection is defined in terms of so-called *multi-items*. For each order block $j \in O'$ we create a multi-item, as illustrated on Figure 5.4. Each multi-item is assigned to exactly one group G_i, $i \in O$, with weight

$$w_j = \begin{cases} \lfloor Z_s/z_j \rfloor - 1 & \text{if } 0 < Z_s \mod z_j < \mu, \\ \lfloor Z_s/z_j \rfloor & \text{otherwise,} \end{cases}$$

and value $p_j w_j$. We have now defined an instance of the restricted two-dimensional knapsack problem, defined in the previous section. The instances for the other projections are defined similarly.

Once these two-dimensional instances have been solved, their solutions are interpreted in terms of the three-dimensional problem.

5.4 Implementation

The algorithmic part of our decision support system was implemented in $C++$, with the exception of the algorithms for the one- and two-dimensional knapsack problems, which were implemented in C. We used CPLEX V4.0 callable library (CPLEX 6.5, 1999) to solve our LP-relaxations, which typically had about one hundred rows and up to several thousand columns. In order to keep the size of our active LP-formulation under control, we used a column pool, which allowed us to let the size of the active formulation vary between $5N$ and $10N$. The active formulation defines the problem which is sent to CPLEX, while the column pool contains the variables which were once generated but were priced negatively in the later solutions. When a number of variables in the active formulation has become larger than $10N$, we move all the variables from the active formulation to the column pool, leaving at most 5 variables per each row, such that they have largest reduced costs, given by (5.7), among all the variables covering this row, i.e. for which the corresponding column has a nonzero entry in this row. Subsequently, we add variables with the largest reduced cost from the column pool to the active formulation, to make its size equal to $5N$. While solving the pricing problem, we first search the column pool for the favorably priced variables. For each variable we calculate an upper bound on its reduced cost using the information stored for each column (see Section 5.3.2). Namely, we calculate the first term in (5.7) exactly, and use the following estimate for the second term: $\sum_{i \in O} a_{ji} \max_{l \geq m} u_l$, where m is the index of the first nonzero entry in a column j. We proceed with generating new columns as described in Section 5.3.3. In case a positively priced column is found, we add at most 5 of them at a time to the active formulation.

The bottleneck of our algorithm is in solving the pricing problem. Therefore, we implemented several modes of pricing, which are controlled by the user. In a *full pricing*, the scheme described in Section 5.3.3 is applied without any limits. On the large instances, the full pricing may take hours of computation. Typically, the full pricing is used during the night runs. A *partial pricing* is used if solution is required within several minutes. In this case we limit the second step of our approximation algorithm for the pricing problem (see the beginning of Section 5.3.3) to only several promising X-slices. To cover the case when solution is required immediately, we implemented options to switch the second step off or to avoid solving the pricing problem altogether. In this way we managed to design a tool which is useful for both on- and off-line planning situations. In our implementation, the user specifies an amount of time available for computations, and the level of pricing is adjusted dynamically using estimates of the remaining running time.

Representation of solution is implemented in the graphical user interface. An output of our algorithm is interpreted by a program written by Michael Beckker and Jan Brands. A solution is represented by a sequence of cutting stages, in the depth-first manner. An example is given in Appendix A.

5.5 Results and conclusions

The system we designed and implemented is being used. Numerous tests indicate that cutting plans generated with our system can be executed without further adjustment.

We performed careful project evaluation with the project group which resulted in the following conclusions.

- Improvement in the organization of the cutting process is the largest positive effect of the project.
- Experiments showed improvement in quality of cutting, but it is difficult to quantify the improvement because of the lack of historical data.
- There is a need for better documentation and support.
- The organization and execution of the project was generally acceptable. Improvements can be made in communication and punctuality of the participants.
- The issue of integration of the system into production process and the attitude of operators towards the system are crucial for the successful exploitation of the system.

Chapter 6

Conclusions

The nature of the research presented in this thesis suggests conclusions about a methodology of decision support by combinatorial optimization. A methodology is a problem oriented procedure or approach which incorporates a particular paradigm. For the purpose of this study, a paradigm is defined as a distinct and, in a certain branch of science, generally accepted way of thinking about problems (Kuhn, 1970). The paradigm of operations research, of which decision support by combinatorial optimization is a part, is generally attributed to Simon (1960) and can be summarized in the following three statements:

- In pursuit of goals, managers take decisions and so solve problems.
- Problems are gaps between performance and goals.
- Problem solving is finding suitable means to achieve goals.

Speaking about a contribution of my work in developing a methodology of decision support by combinatorial optimization, it is convenient to set the results of my thesis in the framework of so-called *action research* (Susman and Evered, 1978; Gilmore et al., 1985; Checkland, 1985). Action research can roughly be described by a two-step procedure:

$\mathcal{A}$. apply a methodology to a real-life situation with the goal to improve this situation;

$\mathcal{B}$. analyze $\mathcal{A}$ with the goal to improve the methodology used.

In my own action research I departed from the methodology of Daellenbach et al. (1983), which incorporates the following five concepts:

1. formulating the problem;
2. constructing a mathematical model;
3. solving the model;
4. deriving a solution to the model;
5. implementing and maintaining the solution.

My experience with the application of this methodology to real-life situations and the analysis of this experience is given in the conclusions to the preceding chapters. Here I directly proceed with describing an improved methodology, based on this experience. I will comment on each of its four phases before discussing the main difference between the two methodologies.

1. Modeling phase
   - organizing a project team;
   - formulating the problem and collecting the data;
   - constructing a mathematical model;
   - analyzing and validating the model.
2. Solution phase
   - solving the mathematical model;
   - analyzing the solution technique;
   - formulating an advice.
3. Implementation phase
   - implementing the advice;
   - verifying and testing the implementation;
   - integrating the implementation into the existing decision process;
   - reporting.
4. Maintenance phase
   - maintaining the model and its solution;
   - maintaining the advice and its implementation.

I make a distinction between four phases of a consultancy project. The phases are typically performed sequentially in practice. It is possible to foresee a situation when loops back are desirable and planned, but I would rather consider such a situation as a sequence of follow-up projects. On the other hand, the activities within each phase can and often should be performed iteratively.

The primary goal of the *modeling phase* is to construct an adequate model of a real-life situation, and support it with the arguments for its validity. A project team is very helpful in accessing the four major components of a problem, as seen from an OR prospective: the decision maker, the objectives, the alternative courses of action, and the environment (Flood and Carson, 1990). The process of constructing a mathematical model, its analysis and validation, are discussed in Chapter 1. Here I only want to reiterate the not-so-obvious idea that the modeling phase should not be reduced to its third activity.

The *solution phase* starts with a given mathematical model and produces an advice, i.e. an indication of what can and what should to be done in the real-life situation being modeled. It is foreseeable that the whole project ends after this phase. Another possible outcome of this phase is the initiation of a new project. However, an emergency loop back to the modeling phase here, usually indicates a failure of the model validation process.

The *implementation phase* is interpreted in a very broad sense here. It may include implementation of soft- or hardware, as well as implementation of an organizational advice. So far, I only had experience with implementation of software in this phase. For that

purpose, I strongly recommend to use a well structured approach, e.g. the unified software development process of Jacobson et al. (1999).

It is important to notice that the *maintenance phase* does not only involve the maintenance of the implementation, it is rather concerned with maintaining the results of all preceding phases, including the model, the solution technique, the advice and the implementation.

The major difference between the proposed methodology and the methodology of Daellenbach et al. (1983) is the emphasis of the former one on the "soft" side of decision support by combinatorial optimization. Here, by the soft side I mean activities which involve both the actual situation and its model, so that the interaction between the two is in focus. Such activities are indeed soft, at least in contrast to such "hard" activities as analyzing a mathematical model, or designing a certain algorithm. The hard activities can be characterized in terms of complexity and optimality, while the soft activities often have to do with concepts like adequateness and rationality.

I am convinced that the strength of decision support by combinatorial optimization is in the hard core of mathematical modeling, which allows one to analyze complex situations and to make a choice between many alternative courses of action. However, the soft side of decision support by combinatorial optimization is necessary in order to make such analysis and choice meaningful.

Appendix A

Example of industrial cutting

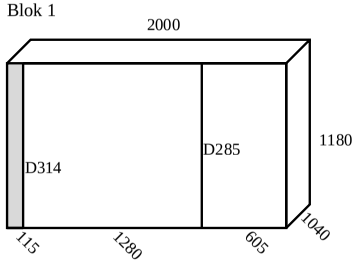
In an instance of the industrial cutting problem at DUMO N.V. there are two stock blocks of sizes 2000mm×1040mm×1180mm and 2460mm×2000mm×1180mm. The order blocks are given in Table A.1. A number of allowed orientations of an order block is given in column *orient.* In this instance all orientations are allowed.

Table A.1: Example of a set of orders

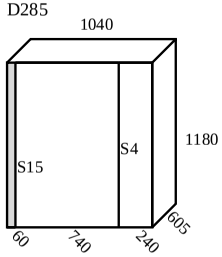
label	order ID	demand	X-size	Y-size	Z-size	orient.
I0	2483/019	1	450	535	580	6
I1	2483/013	1	450	605	120	6
I2	2483/020	1	450	730	370	6
I3	2483/018	1	1460	535	770	6
I4	2483/014	1	740	730	600	6
I5	2483/015	1	1370	600	360	6
I6	2483/017	1	1210	570	260	6
I7	2483/047	1	640	350	350	6
I8	2483/050	1	1250	600	230	6
I9	2483/051	1	1280	1000	410	6
I10	2483/058	1	700	1070	820	6
I11	2483/060	1	830	450	240	6
I12	2483/084	1	200	499	240	6
I13	2483/085	1	200	1214	240	6
I14	2483/086	1	200	1414	350	6
I15	2483/087	1	200	1664	140	6
I16	2483/088	1	1580	600	670	6
I17	2483/090	1	330	600	490	6

A solution to this instance in given on the following 4 pages.

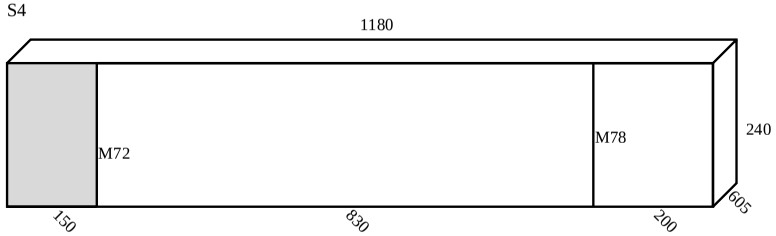
Blok 1



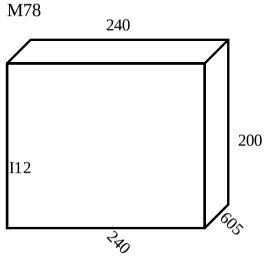
D285



S4

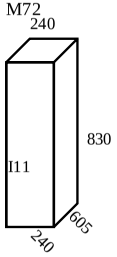


M78



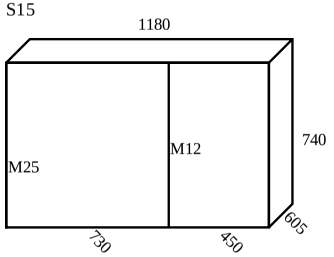
I12: 2483/084, 1x 499mm + extra:106mm; (deelorder klaar.)

M72

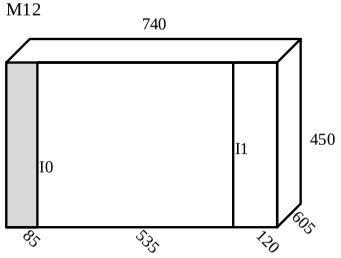


I11: 2483/060, 1x 450mm + extra:155mm; (deelorder klaar.)

S15

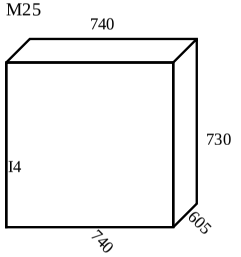


M12



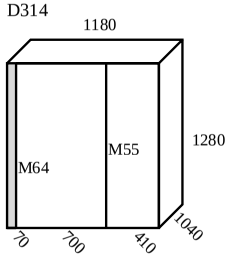
I1: 2483/013, 1x 605mm + extra:0mm; (deelorder klaar.)
I0: 2483/019, 1x 580mm + extra:25mm; (deelorder klaar.)

M25

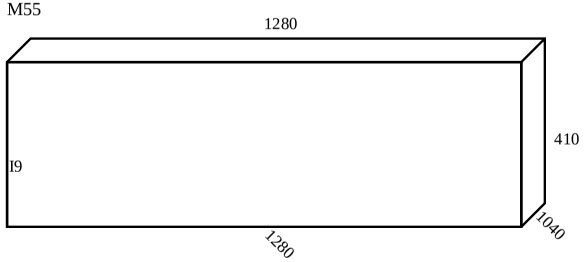


I4: 2483/014, 1x 600mm + extra:5mm; (deelorder klaar.)

D314

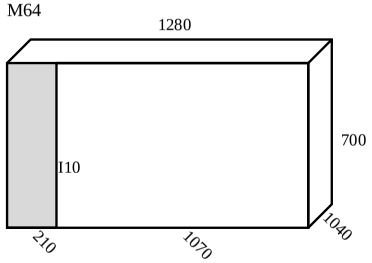


M55



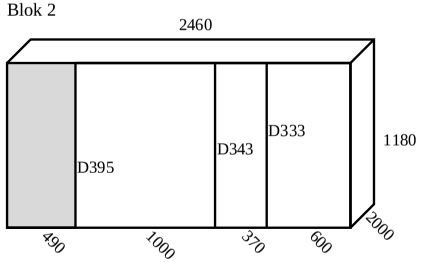
I9: 2483/051, 1x 1000mm + extra:40mm; (deelorder klaar.)

M64

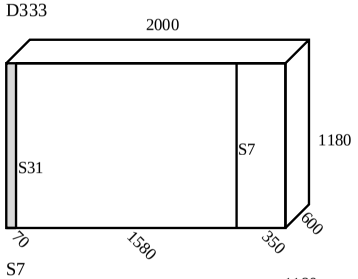


I10: 2483/058, 1x 820mm + extra:220mm; (deelorder klaar.)

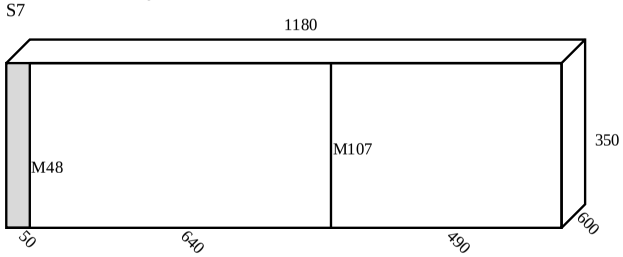
Blok 2



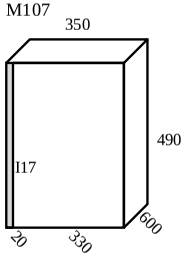
D333



S7

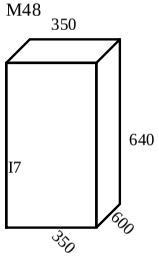


M107



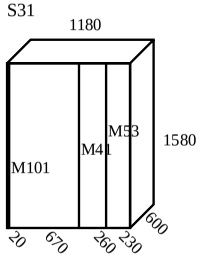
I17: 2483/090, 1x 600mm + extra:0mm; (deelorder klaar.)

M48

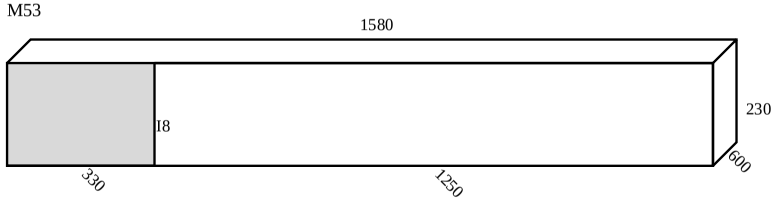


I7: 2483/047, 1x 350mm + extra:250mm; (deelorder klaar.)

S31

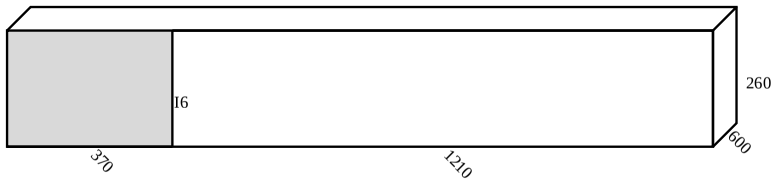


M53



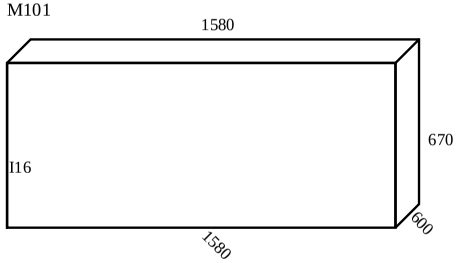
I8: 2483/050, 1x 600mm + extra:0mm; (deelorder klaar.)

M41



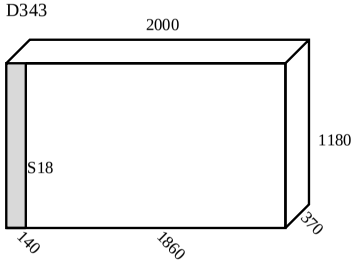
I6: 2483/017, 1x 570mm + extra:30mm; (deelorder klaar.)

M101

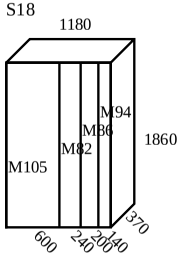


I16: 2483/088, 1x 600mm + extra:0mm; (deelorder klaar.)

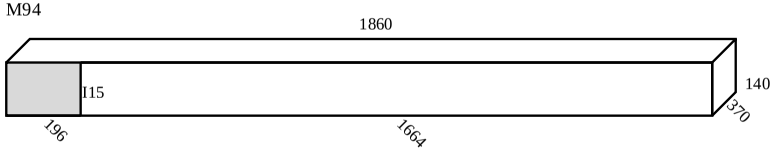
D343



S18

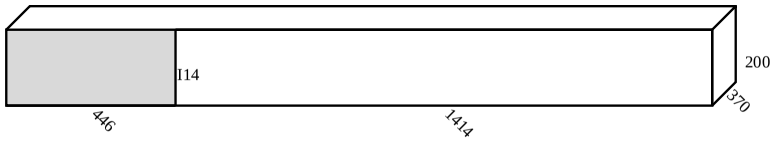


M94



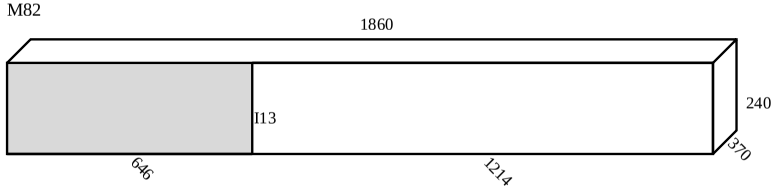
I15: 2483/087, 1x 200mm + extra:170mm; (deelorder klaar.)

M86



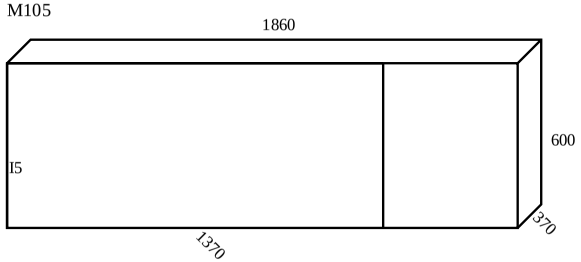
I14: 2483/086, 1x 350mm + extra:20mm; (deelorder klaar.)

M82



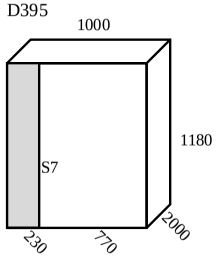
I13: 2483/085, 1x 200mm + extra:170mm; (deelorder klaar.)

M105

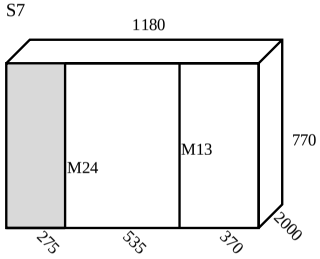


I5: 2483/015, 1x 360mm + extra:10mm; (deelorder klaar.)

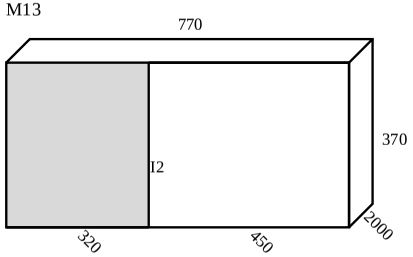
D395



S7

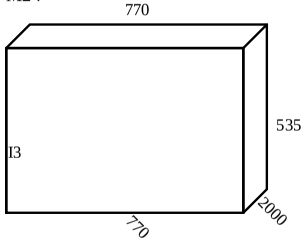


M13



I2: 2483/020, 1x 730mm + extra:1270mm; (deelorder klaar.)

M24



I3: 2483/018, 1x 1460mm + extra:540mm; (deelorder klaar.)

Bibliography

Aardal, K., C. A. J. Hurkens, J. K. Lenstra, and S. R. Tiourine (1996). Algorithms for frequency assignment problems. *CWI Quarterly 9*, 1–8.

Aardal, K. and C. P. M. Van Hoesel (1996). Polyhedral techniques in combinatorial optimization I: Theory. *Statistica Neerlandica 50*, 4–26. Also available at `http://www.cs.ruu.nl:80/people/aardal/survey1.ps`.

Aardal, K. and C. P. M. Van Hoesel (1999). Polyhedral techniques in combinatorial optimization II: Computations. *Statistica Neerlandica*, to appear. Also available at `http://www.cs.ruu.nl:80/people/aardal/survey2.ps`.

Aardal, K., C. P. M. Van Hoesel, A. Hipolito, and B. Jansen (1998). Modeling and solving a frequency assignment problem. Technical report, Department of Computer Science, Utrecht University.

Aarts, E. H. L. and J. H. M. Korst (1989). *Simulated Annealing and Boltzmann Machines: a Stochastic Approach to Combinatorial Optimization and Neural Networks*. Wiley, Chichester.

Aarts, E. H. L. and J. K. Lenstra (eds.) (1997). *Local Search in Combinatorial Optimization*. Wiley, Chichester.

Ackoff, R. L. (1979). The future of operations research is past. *Journal of the Operational Research Society 30*, 93–104.

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (1993). *Network Flows: Theory and Applications*. Prentice Hall, Englewood Cliffs.

Alizadeh, F. (1995). Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization 5*, 13 – 51.

Ascheuer, N., M. Grötschel, N. Kamin, and J. Rambau (1998). Combinatorial online optimization in practice. *OPTIMA - Mathematical Programming Society Newsletter 57*, 1–6. Also available at `http://www.zib.de/ascheuer/publications.html`.

Ascheuer, N., M. Grötschel, S. O. Krumke, and J. Rambau (1998). Combinatorial online optimization. In *Proceedings of the International Conference of Operations Research Zürich (OR'98)*, pp. 21–37. Springer, Berlin. Also available at `http://www.zib.de/ascheuer/publications.html`.

Balas, E. (1969). Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations Research 17*, 941–957.

Balas, E. and P. Toth (1985). Branch and bound methods. In E. L. Lawler, J. K. Lenstra,

A. H. G. Rinnooy Kan, and D. B. Shmoys (eds.), *The Traveling Salesman Problem*, pp. 361–401. Wiley, Chichester.

Balas, E. and A. Vazacopoulos (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science 44*, 262–275.

Barnett, A. (1994). Models fail. In S. M. Pollock, M. H. Rothkopf, and A. Barnett (eds.), *Operations Research and the Public Sector, Handbooks in Operations Research and Management Science*, pp. 47–66. North-Holland, Amsterdam.

Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance (1998). Branch-and-price: column generation for solving huge integer programs. *Operations Research 46*, 316–329.

Bellman, R. E. and S. E. Dreyfus (1962). *Applied Dynamic Programming*. Princeton University Press, Princeton.

Bensana, E. and T. Schiex (1995). Partial constraint satisfaction. Technical Report CALMA 2.2.3, Centre d'Etudes et de Recherches de Toulouse, France.

Berge, C. (1976). *Graphs and Hypergraphs*. North-Holland, Amsterdam.

Bertsekas, D. P. (1998). *Network Optimization: Continuous and Discrete Models*. Athena Scientific, Belmont.

Bitner, J. R., G. Ehrlich, and E. M. Reingold (1976). Efficient generation of the binary reflected Gray code and its applications. *Communications of the ACM 9*, 517–521.

Bouju, A., J. F. Boyce, C. H. D. Dimitropoulos, G. Vom Scheidt, and J. G. Taylor (1994). Tabu search and GENET. Technical Report CALMA 2.3.4, Centre d'Etudes et de Recherches de Toulouse, France. Also available at `http://www.win.tue.nl/~wscor/calma.html`.

Bourret, P. (1995). Simulated annealing. Technical Report CALMA 2.3.1, Centre d'Etudes et de Recherches de Toulouse, France.

Brélaz, D. (1973). New methods to color the vertices of a graph. *Communications of the ACM 16*, 575–577.

Caprara, A., M. Fischetti, P. Toth, D. Vigo, and P. L. Guida (1997). Algorithms for railway crew management. *Mathematical Programming 79*, 125–141.

Castelino, D. J., S. Hurly, and N. M. Stephens (1996). A tabu search algorithm for frequency assignment. *Annals of Operations Research 63*, 301–319.

Chardaire, P., A. Kapsalis, J. W. Mann, V. J. Rayward-Smith, and G. D. Smith (1995). Applications of genetic algorithms in telecommunications. In J. Alspector, R. Goodman, and T. X. Brown (eds.), *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunication 2*, pp. 290–299. Lawrence Erlbaum, Hillsdale.

Checkland, P. B. (1985). From optimizing to learning: a development of systems thinking for the 1990s. *Journal of the Operational Research Society 36*, 757–767.

Chvátal, V. (1983). *Linear Programming*. Freeman, New York.

Clarke, E. M. and J. M. Wing (1996). Formal methods: State of the art

and future directions. *ACM Computing Surveys 28*, 626–643. Also available at `http://www.cs.cmu.edu/~wing/`.

Constantinides, G. M. and A. G. Malliaris (1995). Finance. In R. A. Jarrow, V. Maksimovic, and W. T. Ziemba (eds.), *Operations Research and the Public Sector, Handbooks in Operations Research and Management Science*, p. 18. North-Holland, Amsterdam.

Corbett, C. J., W. J. A. M. Overmeer, and L. N. Van Wassenhove (1995). Strands of practice in OR (the practitioner's dilemma). *European Journal of Operational Research 87*, 484–499.

CPLEX 6.5 (1999). *Documentation Supplement*. ILOG Inc., CPLEX Division, Incline Village.

Daellenbach, H. G., J. A. George, and D. C. McNickle (1983). *Introduction to Operations Research Techniques*. Allyn & Bacon, Boston.

Dantzig, G. B. and M. N. Thapa (1997). *Linear Programming 1: Introduction*. Springer, New York.

De Waal, A. G. and L. C. R. J. Willenborg (1995). Optimum global recoding and local suppression. Technical report, Statistics Netherlands, Voorburg, The Netherlands.

De Waal, A. G. and L. C. R. J. Willenborg (1998). Optimum local suppression in microdata. *Journal of Official Statistics, 14*, 421–436.

Denardo, E. V. (1982). *Dynamic Programming Models and Applications*. Prentice Hall, Englewood Cliffs.

Dimitropoulos, C. H. D. (1998). Private communication. King's College London, email: `hcd@phalcon.ph.kcl.ac.uk`.

Einstein, A. (January, 1921). Geometry and experience, address to Prusian Academy of Sciences, Berlin. In *Sidelights on Relativity, Mathuen, London, 1922*, p. 28.

Emshoff, J. R. and R. L. Sisson (1970). *Design and Use of Computer Simulation Models*. Macmillan, New York.

Fisher, H. and G. L. Thompson (1963). Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson (eds.), *Industrial Scheduling*, pp. 225–251. Prentice-Hall, Englewood Cliffs.

Fisher, M. L. (1981). The Lagrangean relaxation method for solving integer programming problems. *Management Science 27*, 1–18.

Flood, R. L. and E. R. Carson (1990). *Dealing with Complexity*. Plenum Press, New York.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: a Guide to the Theory of NP-completeness*. Freeman, New York.

Gass, S. I. (1983). Decision-aiding models: validation, assessment and related issues for policy analysis. *Operations Research 31*, 603–631.

Geoffrion, A. M. (1974). Lagrangean relaxation for integer programming. *Mathematical Programming Study 2*, 82–114.

Gilmore, P. C. and R. E. Gomory (1961). A linear programming approach to the cutting-stock problem. *Operations Research 9*, 849–859.

Gilmore, P. C. and R. E. Gomory (1965). Multistage cutting stock problems of two and more dimensions. *Operations Research 13*, 94–120.

Gilmore, T., J. Krantz, and R. Ramirez (1985). Action based modes of inquiry and the host-researcher relationship. *Consultation 5*, 160–176.

Glover, F. (1989). Tabu search - part I. *ORSA Journal on Computing 1*, 190–206.

Glover, F. (1990). Tabu search - part II. *ORSA Journal on Computing 2*, 4–32.

Glover, F. and M. Laguna (1997). *Tabu Search*. Kluwer Academic Publishers, Boston.

Goemans, M. X. (1997). Semidefinite programming in combinatorial optimization. *Mathematical Programming 79*, 143 – 161.

Golden, B. L. and W. R. Stewart (1985). Empirical analysis of heuristics. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (eds.), *The Traveling Salesman Problem*, pp. 207–250. Wiley, Chichester.

Ha, T. T. (1990). *Digital Satellite Communications*. McGraw-Hill, New York.

Hadjiconstantinou, E. and N. Christofides (1995). An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operational Research 83*, 39–56.

Hadley, G. (1964). *Nonlinear and Dynamic Programming*. Addison-Wesley, Reading.

Haessler, R. W. and P. E. Sweeney (1991). Cutting stock problems and solution procedures. *European Journal of Operational Research 54*, 141–150.

Hajema, W., M. Minoux, and C. West (1993). Statement of the radio link frequency assignment problem. Technical report, Request for proposals on the CALMA project.

Hale, W. K. (1980). Frequency assignment: Theory and applications. *Proceedings of the IEEE 68*, 1497–1514.

Hao, J.-K., R. Dorne, and P. Galinier (1998). Tabu search for frequency assignment in mobile radio networks. *Journal of Heuristics 4*, 47–62.

Hax, A. C. and D. Candea (1984). *Production and Inventory Management*. Prentice Hall, Englewood Cliffs.

Held, M. and R. M. Karp (1970). The traveling-salesman problem and minimum spanning trees. *Operations Research 18*, 1138–1162.

Held, M. and R. M. Karp (1971). The traveling-salesman problem and minimum spanning trees: part ii. *Mathematical Programming 1*, 6–25.

Held, M., P. Wolfe, and H. P. Crowder (1974). Validation of subgradient optimization. *Mathematical Programming 6*, 62–88.

Hermann, C. (1967). Validation problems in games and simulations. *Behavioral Science 12*, 216–230.

Hurkens, C. A. J. and S. R. Tiourine (1998). Models and methods for the microdata protection problem. *Journal of Official Statistics, 14*, 437–448.

ILOG (1997). Optimization technology white paper: a comparative study of optimization techniques. Technical report, ILOG, Inc. Also available at `http://www.ilog.fr/products/optimization/tech_papers.cfm`.

Jacobson, I., G. Booch, and J. Rumbaugh (1999). *The Unified Software Development Process*. Addison-Wesley, Amsterdam.

Johnson, D. S. and L. A. McGeoch (1997). The traveling salesman problem: a case study. In E. H. L. Aarts and J. K. Lenstra (eds.), *Local Search in Combinatorial Optimization*, pp. 215–310. Wiley, Chichester.

Johnson, D. S. and C. H. Papadimitriou (1985). Performance guarantees for heuristics. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (eds.), *The Traveling Salesman Problem*, pp. 145–180. Wiley, Chichester.

Jünger, M., G. Reinelt, and S. Thienel (1995). Practical problem solving with cutting plane algorithms in combinatorial optimization. In W. Cook, L. Lovász, and P. D. Seymour (eds.), *Combinatorial Optimization*, Volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 111–152. American Mathematical Society, Providence.

Kall, P. and S. W. Wallace (1994). *Stochastic Programming*. Wiley, Chichester.

Kamath, A. P., N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende (1990). Computational experience with an interior point algorithm on the satisfiability problem. *Annals of Operations Research 25*, 43–58.

Karmarkar, N. K., M. G. C. Resende, and K. G. Ramakrishnan (1991). An interior point algorithm to solve computationally difficult set covering problems. *Mathematical Programming 52*, 597–618.

Karp, R. M. and J. M. Steele (1985). Probabilistic analysis of heuristics. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (eds.), *The Traveling Salesman Problem*, pp. 181–206. Wiley, Chichester.

Kernighan, B. W. and S. Lin (1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal 49*, 291–307.

Keys, P. (1991). *Operational Research and Systems : the Systemic Nature of Operational Research*. Plenum Press, New York.

Kleijnen, J. P. C. and G. A. Alink (1992). Verification of simulation models: Minehunting case study. Technical Report FEW 537, Department of Economics, Tilburg University, The Netherlands.

Kleijnen, J. P. C. and W. Van Groenendaal (1992). *Simulation: a Statistical Perspective*. Wiley, Chichester.

Kolen, A. W. J. (1997). A genetic algorithm for the partial binary constraint satisfaction problem: An application to a frequency assignment problem. Working paper, Maastricht University.

Kolen, A. W. J. and C. P. M. Van Hoesel (1995). Constraint satisfaction. Technical Report CALMA 2.2.2, Centre d'Etudes et de Recherches de Toulouse, France. Also available at `http://www.win.tue.nl/~wscor/calma.html`.

Koster, A. M. C. A. (November, 1999). *Frequency Assignment: Models and Algorithms.* Ph. D. thesis, Maastricht university.

Koster, A. M. C. A. and C. P. M. Van Hoesel (1998). The frequency assignment problem: lower bound and optimal solutions via tree decomposition. Maastricht university, email: `A.Koster@ke.unimaas.nl`.

Kuhn, T. S. (1970). *The Structure of Scientific Revolutions* (Second ed.). University of Chicago Press.

Lanfear, T. A. (1989). Graph theory and radio frequency assignment. Technical report, Allied Radio Frequency Agency, NATO Headquarters, Brussels.

Law, A. M. and W. D. Kelton (1991). *Simulation Modeling and Analysis.* McGraw-Hill, London.

Lawler, E. L. and D. E. Wood (1966). Branch-and-bound methods: a survey. *Operations Research 14*, 699–719.

Lee, W. C. Y. (1995). *Mobile Cellular Telecommunications: Analog and Digital Systems.* McGraw-Hill, New York.

Lenstra, J. K. and A. H. G. Rinnooy Kan (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics 4*, 121–140.

Lin, S. and B. W. Kernighan (1973). An effective heuristic algorithm for the traveling salesman problem. *Operations Research 21*, 498–511.

Linderoth, J. T. and M. W. P. Savelsbergh (1999). A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, to appear. Also available at `http://udaloy.isye.gatech.edu/~mwps`.

Lustig, I. J., R. E. Marsten, and D. F. Shanno (1994). The last word on interior point methods for linear programming - for now. *ORSA Journal on Computing 6*, 35–36.

Mackworth, A. K. (1987). Constraint satisfaction. In S. Shapiro (ed.), *Encyclopedia of Artificial Intelligence*, pp. 205–211. Wiley, New York.

Martello, S. and P. Toth (1990). *Knapsack Problems: Algorithms and Computer Implementations.* Wiley, Chichester.

Martin, P. and D. B. Shmoys (June, 1996). A new approach to computing optimal schedules for the job-shop scheduling problem. In *Proceedings of the 5th International IPCO Conference, Vancouver, British Columbia, Canada*, pp. 389–403.

Midgley, G. (1998). Theory and practice in operations research. *Journal of the Operational Research Society 49*, 1219–1220.

Minoux, M. (1986). *Mathematical Programming: Theory and Algorithms.* Wiley, Chichester.

Montgomery, D. G. and G. C. Runder (1999). *Applied Statistics and Probability for Engineers.* Wiley, New York.

Murphy, F. H. (1998). The occasional observer: some simple precepts for project success. *Interfaces 28*, 25–28.

Nemhauser, G. L., M. W. P. Savelsbergh, and G. C. Sigismondi (1994). MINTO, a

Mixed INTeger Optimizer. *Operations Research Letters 15*, 47–58. Also available at `http://udaloy.isye.gatech.edu/~mwps`.

Nemhauser, G. L. and L. A. Wolsey (1988). *Integer and Combinatorial Optimization*. Wiley, New York.

Nowicki, E. and C. Smutnicki (1996). A fast taboo search algorithm for the job shop problem. *Management Science 42*, 797–813.

Padberg, M. W. (1973). On the facial structure of set packing polyhedra. *Mathematical Programming 5*, 199–215.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley, Reading.

Papadimitriou, C. H. and K. Steiglitz (1998). *Combinatorial Optimization: Algorithms and Complexity*. Dover, Mineola.

Pasechnik, D. V. (1998). An interior point approximation algorithm for a class of combinatorial optimization problems: implementation and enhancements. Technical report, Delft University of Technology.

Pollock, S. M. and M. D. Maltz (1994). Operations research in the public sector: an introduction and a brief history. In S. M. Pollock, M. H. Rothkopf, and A. Barnett (eds.), *Operations Research and the Public Sector, Handbooks in Operations Research and Management Science*, p. 3. North-Holland, Amsterdam.

Rayward-Smith, V. J., G. McKeown, G. D. Smith, A. W. J. Kolen, P. Chardaire, A. Kapsalis, and S. Rush (1995). Genetic algorithms. Technical Report CALMA 2.1.6, Centre d'Etudes et de Recherches de Toulouse, France.

Roberts, F. R. (1989). T-colorings of graphs: recent results and open problems. *Discrete Mathematics 93*, 229–245.

Robertson, N. and P. D. Seymour (1986). Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms 7*, 309–322.

Roy, B. and B. Sussman (1964). Les problèmes d'ordonnancement avec contraintes disjonctives. Note DS No. 9 bis, SEMA, Paris.

Savelsbergh, M. W. P. (1994). Preprocessing and probing techniques for mixed integer programming. *ORSA Journal on Computing 6*, 445–454.

Simon, H. A. (1960). *The New Science of Management Decision*. Harper & Row, New York.

Stougie, L. and M. H. Van der Vlerk (1997). Stochastic integer programming. In M. DellÁmico, F. Maffioli, and S. Martello (eds.), *Annotated Bibliographies in Combinatorial Optimization*, pp. 127–141. Wiley, Chichester.

Susman, G. and R. D. Evered (1978). An assessment of the scientific merits of action research. *Administrative Science Quarterly 23*, 582–603.

Thienel, S. (December, 1995). *ABACUS - A Branch-And-CUt System*. Ph. D. thesis, Universität zu Köln, D 50969, Köln. Also available at `http://www.informatik.uni-koeln.de/ls_juenger/projects/abacus.html`.

Tilanus, C. B. (1985). Failures and successes of quantitative methods in management. *European Journal of Operational Research 19*, 170–175.

Tiourine, S. R., C. A. J. Hurkens, and J. K. Lenstra (1999). Local search algorithms for the radio link frequency assignment problem. *Telecommunication systems, to appear.*

Todd, M. J. (1994). Theory and practice for interior-point methods. *ORSA Journal on Computing 6*, 28–31.

Tsang, E. P. K. (1993). *Foundations of Constraint Satisfaction.* Academic Press, London.

Tsang, E. P. K. and C. J. Wang (1992). A generic neural network approach for constraint satisfaction problems. In J. G. Taylor (ed.), *Neural Network Applications*, pp. 12–22. Springer-Verlag.

Tuttlebee, W. H. W. (ed.) (1990). *Cordless Telecommunications in Europe: the Evolution of Personal Communications.* Springer, Berlin.

Vaessens, R. J. M., E. H. L. Aarts, and J. K. Lenstra (1996). Job shop scheduling by local search. *INFORMS Journal on Computing 8*, 302–317.

Van der Bruggen, L. J. J., J. K. Lenstra, and P. C. Schuur (1993). Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science 27*, 298–311.

Van Hoesel, C. P. M., J. K. Lenstra, S. R. Tiourine, and B. Veltman (1993). Modelbeschrijving. Technical Report IWDE 93-09, Eindhoven University of Technology, the Netherlands.

Van Laarhoven, P. J. M., E. H. L. Aarts, and J. K. Lenstra (1992). Job shop scheduling by simulated annealing. *Operations Research 40*, 113–125.

Vom Scheidt, G. (1995). Extension of GENET to partial constraint satisfaction problem and constraint satisfaction optimization problems. Master's thesis, King's College London.

Warners, J. P. (1998). A nonlinear approach to a class of combinatorial optimization problems. *Statistica Neerlandica 52*, 162–184.

Warners, J. P. (September, 1999). *Nonlinear Approaches to Satisfiability Problems.* Ph. D. thesis, Eindhoven University of Technology.

Warners, J. P., T. Terlaky, C. Roos, and B. Jansen (1997a). Potential reduction algorithms for structured combinatorial optimization problems. *Operations Research Letters 21*, 55–64.

Warners, J. P., T. Terlaky, C. Roos, and B. Jansen (1997b). A potential reduction approach to the frequency assignment problem. *Discrete Applied Mathematics 78*, 251–282.

Willenborg, L. C. R. J. (1997). Personal communication.

Willenborg, L. C. R. J. and A. G. De Waal (1996). *Statistical Disclosure Control in Practice. Lecture notes on statistics* **111**. Springer, New York.

Willenborg, L. C. R. J. and A. Hundepool (1998). ARGUS for statistical disclosure control. In *Proceedings of the statistical data protection conference, Lisbon, March 25–27, 1998.*

Williams, H. P. (1978). *Model Building in Mathematical Programming.* Wiley, Chichester.

Yeh, R. T., M. M. Tanik, W. Rossak, F. Cheng, and P. A. Ng (1992). Software engineering. In E. G. Coffman, J. K. Lenstra, and A. H. G. Rinnooy Kan (eds.), *Computing, Handbooks in Operations Research and Management Science*, pp. 195–246. North-Holland, Amsterdam.

Author index

Curriculum vitae

Sergey Tiourine was born on December 12th, 1966 in St.-Petersburg, Russia (in Leningrad, the former Soviet Union, if you prefer). He studied astronomy at St.-Petersburg State University from 1984 to 1991, with a break for military service. In the period from 1992 to 1994 he completed the master program "Mathematics for Industry" at Eindhoven University of Technology, the Netherlands. Five years later he obtained his Ph.D. degree at the same university. Currently Sergey is employed by Carmen Systems AB in Göteborg, Sweden.