Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

ALEXANDER MARTIN

Integer Programs with Block Structure

Preprint SC 99-03 (Februar 1999)

Alexander Martin

$\boxed{\text{Integer}}$ $\boxed{\text{Programs}}$ $\boxed{\text{with}}$ $\boxed{\text{Block Structure}}$

Habilitationsschrift
Technische Universität Berlin
Juni 1998 (überarbeitet Februar 1999)

Acknowledgments

At this point I would like to thank some people who accompanied my way to this thesis during the last six years.

First of all, my special thanks go to *Martin Grötschel* who provided me with a very nice working atmosphere at ZIB. He gave me all the freedom for my own research and still supplied me with all his support. I am also very grateful to *Bob Bixby*. We became friends over the last years. He was the one who understood my needs and desires during the most difficult days and helped me to follow the right path. I also cannot thank *Robert Weismantel* enough for his collaboration and friendship. We had lots of lively discussions on various research and non-research topics that pushed me in the right direction.

In particular, I would like to thank all my coauthors for a wonderful cooperation and their permission to use our joint work here. Chapter 2 is joint work with Carlos Ferreira and Robert Weismantel and is published in Ferreira, Martin, and Weismantel [1996]. The contents of Chapter 3 is a summary of papers together with Martin Grötschel and Robert Weismantel (see Grötschel, Martin, and Weismantel [1995], and [1996a, 1996b, 1996c, 1996d, 1997]). Chapter 4 is joint work with Geir Dahl and Mechthild Stoer and is published in Dahl, Martin, and Stoer [1995]. Chapter 5 is partially together with Robert Weismantel, in particular Section 5.4 that appeared as a preprint (see Martin and Weismantel [1997]). Chapter 6 is jointly with Ralf Borndörfer and Carlos Ferreira and is published in Borndörfer, Ferreira, and Martin [1998]. Finally, Chapter 7 is joint work with Bob Bixby, see Bixby and Martin [1995], and Chapter 8 with Robert Weismantel, see Martin and Weismantel [1998]. To all these people, to my colleagues and friends Ralf Borndörfer and Andreas Löbel, and to our secretaries Bettina Kasse und Sybille Mattrisch I express my deepest thanks for their support, for reading, criticism, and all the inspiring discussions on various topics of this thesis.

I gratefully acknowledge the support of the Konrad-Zuse-Zentrum für Informationstechnik Berlin, the support by the German Academic Exchange Service (DAAD) within the project PROBRAL, the support by Telenor Research, Norway, and by the Center for Research on Parallel Computing (CRPC), Houston, Texas.

Last, but not least my warmest thanks are to my wife *Gabriele*. She constantly took care of all my needs and gave me all the support I needed. She and our daughter Alina build the home from which I drew the mental power for my profession. Thank you, Gabi, I dedicate this thesis to you.

Berlin, June 1998 Alexander Martin

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The study and solution of linear integer programs lies in the heart of discrete optimization. Many different kinds of problems in science, technology, business, and society can be modeled as linear integer programming problems. It is not in the least surprising that there is no unique method that solves all integer programming problems. Among the most successful methods are currently LP based branch-and-bound algorithms where the underlying linear programs (LPs) are possibly strengthened by cutting planes. For example, most commercial integer programming solvers or special purpose codes for problems like the traveling salesman problem are based on this method.

In this thesis we study and solve integer programs with block structure, i. e., problems that after the removal of certain rows (or columns) of the constraint matrix decompose into independent subproblems. The matrices associated with each subproblem are called blocks and the rows (columns) to be removed linking constraints (columns). Integer programs with block structure come up in a natural way in many real-world applications. The blocks often model certain objects or decisions in a division of a company, in a technical unit, or in a time period. These individual blocks are linked by a couple of constraints that model the fact that the objects must share certain resources or that model possible interactions between the decisions that cover the whole company, technical process, or time horizon. Just to name some concrete examples, the applications range from vehicle scheduling, see Löbel [1997], over problems in telecommunication, see Section 4, to stochastic integer programming, see Stougie and van der Vlerk [1997] or Carøe, Ruszczyński, and Schultz [1997] for a recent application.

The methods that are widely used to tackle integer programs with block structure are decomposition methods. The idea is to decouple the linking constraints (variables) from the problem and treat them at a superordinate level, often called master problem. The resulting residual subordinate problem then decomposes into independent subproblems that often can be solved more efficiently. Decomposition methods now work alternately on the master and subordinate problem and iteratively exchange information to solve the original problem to optimality. Two well known examples of this approach are Dantzig-Wolfe decomposition (Dantzig and Wolfe [1960]) and Benders decomposition (Benders [1962]). Soumis [1997] gives a survey on different decomposition methods for linear and integer programs. We remark that decomposition methods are not only used to solve integer programs. They also play a prominent role in combinatorial optimization, for example in the solution of certain graph theoretic problems (see, for instance, Robertson and Seymour [1990]) or of problems in connection with matroids (see, for instance, Truemper [1992]).

In Part I we follow a different approach. We treat the integer programming problem as a whole and keep the linking constraints in the formulation. We consider the associated polyhedra and investigate the polyhedral consequences of the involved linking constraints. The variety and complexity of the new inequalities that come into play is illustrated on three different types of real-world problems. The applications arise in the design of electronic circuits, in telecommunication and production planning. We develop a branch-and-cut algorithm for each of these problems, and our computational results show the benefits and limits of the polyhedral approach to solve these real-world models with block structure.

Part II of the thesis deals with general mixed integer programming problems, that is integer programs with no apparent structure in the constraint matrix. We will discuss in Chapter 5 the main ingredients of an LP based branch-and-bound algorithm for the solution of general integer programs. Chapter 6 then asks the question whether general integer programs decompose into certain block structures and investigate whether it is possible to recognize such a structure. The remaining two chapters exploit information about the block structure of an integer program. In Chapter 7 we parallelize parts of the dual simplex algorithm, the method that is commonly used for the solution of the underlying linear programs within a branch-and-cut algorithm. In Chapter 8 we try to detect small blocks in the constraint matrix and to derive new cutting planes that strengthen the integer programming formulation. These inequalities may be associated with the intersection of several knapsack problems. We will see that they significantly improve the quality of the general integer programming solver introduced in Chapter 5.

Part I

Real-World Models with Block Structure

In the first part of this thesis we discuss three different real-world applications whose mathematical formulations give rise to integer programs with block structure. With these integer programs are associated certain polyhedra and our main focus will be the study of these polyhedra. In particular, we address the following questions:

Do the inequalities associated with the individual polyhedra, i. e., the polyhedra resulting from the single blocks, yield strong or even facet-defining inequalities for the *packing polyhedron*, i. e., the polyhedron corresponding to the original problem? What kind of new inequalities, inequalities that join/combine individual blocks, come into play? Are these *joint inequalities* tractable in the sense that they can be efficiently separated? Are these inequalities helpful within a branch-and-cut algorithm to solve practical problem instances?

It would be bold to claim that we can give answers to these questions for any integer program with block structure. What we want to do however is to (partially) answer these questions for three particular models arising from real-world problems. We will see that the associated integer programs inherit well known structures that play an important role in polyhedral combinatorics: the knapsack problem and the set packing, -partitioning, and -covering problem.

In the first application we will discuss, the *multiple knapsack problem*, the blocks consist of knapsack problems and the linking constraints are set packing and set partitioning inequalities, respectively. The second model, the *Steiner tree packing problem*, has also set packing constraints in the linking part, whereas the blocks turn out to be of set covering type. The third is somehow a combination of the first two models, the blocks consist of set covering constraints and the linking constraints are knapsack inequalities which in turn are linked by set packing constraints.

The real-world applications that lead to these integer programming models range from the design of electronic circuits and super computers, over production planning problems to network design problems in telecommunication.

The outline of the forthcoming three chapters is as follows: In the introductory sections we outline the applications that give rise to the particular integer programs with block structure and present the mathematical formulations. We then present briefly some basic results of the associated polyhedra followed by a discussion of the individual inequalities, i. e., the inequalities resulting from the individual blocks . The main section in each chapter is devoted to *joint inequalities*, inequalities that combine the individual blocks. Finally, we summarize our computational experiences with a branch-and-cut algorithm that are based on the described inequalities. We will point out various difficulties that arise when turning these inequalities into an algorithmic tool and show what problem sizes are solvable with such an approach.

We will not present all the details in each of these sections/chapters. We refrain in particular from giving all the sometimes long and very technical proofs. Details hereto may be found in the cited literature. We rather want to emphasize the difficulties that come up in trying to understand the associated packing polyhedra, and give an impression to what extent this understanding might help in solving practical problems.

Chapter 2

The Multiple Knapsack Problem

2.1 Introduction

In this chapter we study integer programs with block structure where the blocks result from knapsack problems and the linking constraints are of set packing type. We will encounter the knapsack problem and its associated polytope in different sections of this thesis, see, beyond this chapter, Sections 4.3.1 and 5.4 or Chapter 8. Here, we discuss a canonical generalization of the (single) knapsack problem, where instead of just one knapsack a set M of knapsacks is available to which a given set of items can be assigned to. This problem, called the *(weighted) multiple knapsack problems*, is defined as follows: Given a set N of items with weights $f_i > 0$, $i \in N$, a set M of *knapsacks* with capacities $F_k > 0$, $k \in M$, and an objective function $c_{ik}, i \in N, k \in M$, which reflects the profit if item i is assigned to knapsack k. The task is to find an assignment of a subset of the set of items to the set of knapsacks that yields maximum profit.

We denote by $\mathrm{MK}_w\,(N, M, f, F, c)$ an *instance of the weighted multiple knapsack problem*, i. e., a set N of items, a set M of knapsacks, a weight vector $f = (f_i)_{i\in N}$, a capacity vector $F = (F_k)_{k\in M}$, and an objective function $c \in \mathbb{R}^{N\times M}$. We introduce variables $x \in \mathbb{R}^{N\times M}$ with the interpretation $x_{ik} = 1$ if item i is assigned to knapsack k and $x_{ik} = 0$ otherwise. The weighted multiple knapsack problem can be formulated as the following integer program.

$$
\begin{array}{lrll}
 & \max & \displaystyle\sum_{i\in N}\sum_{k\in M} c_{ik}x_{ik} & \\
 & (i) & \displaystyle\sum_{i\in N} f_i x_{ik} \le F_k, & \text{for all } k \in M; \\
(2.1) & (ii) & \displaystyle\sum_{k\in M} x_{ik} \le 1, & \text{for all } i \in N; \\
 & (iii) & 0 \le x_{ik} \le 1, & \text{for all } i \in N, k \in M; \\
 & (iv) & x_{ik} \in \{0,1\}, & \text{for all } i \in N, k \in M.
\end{array}
$$

The constraints (2.1) (i) are called *knapsack inequalities*, the constraints (2.1) (ii) *SOS (Special Ordered Set) inequalities*, and those in (2.1) (iii) *trivial inequalities*. Figure 2.1 illustrates the structure of the constraint matrix of (2.1).

The *multiple knapsack polytope* $P_{MK}\,(N \times M, f, F)$ is defined as the convex hull

$$
\begin{array}{cccccccccccccccccc}
k & k & k & k & k & k & & & & & & & & & & & & \\
 & & & & & & k & k & k & k & k & k & & & & & & \\
 & & & & & & & & & & & & k & k & k & k & k & k \\
p & & & & & & p & & & & & & p & & & & & \\
 & p & & & & & & p & & & & & & p & & & & \\
 & & p & & & & & & p & & & & & & p & & & \\
 & & & p & & & & & & p & & & & & & p & & \\
 & & & & p & & & & & & p & & & & & & p & \\
 & & & & & p & & & & & & p & & & & & & p
\end{array}
$$

Figure 2.1: Structure of the multiple knapsack integer program with $|M| = 3$ and $|N| = 6$: The letter 'k' shows a non-zero entry in a knapsack constraint, the letter 'p' indicates a non-zero entry in an SOS constraint, which is a special set packing constraint.

of all feasible solutions of the multiple knapsack problem that is

$$
(2.2) \qquad P_{\mathrm{MK}}\,(N \times M, f, F) = \mathrm{conv}\{x \in \mathbb{R}^{N\times M} \,:\, x \text{ satisfies } (2.1)\ (i) - (iv)\}.
$$

Before investigating the multiple knapsack polytope in detail, let us point out the applications that motivated the study of this polytope. The problems we have in mind arise in three different applications, namely in the layout of electronic circuits, in the design of processors for main frame computers and in the sugar cane alcohol production in Brazil.

The first problem arises in the (global) design of a main frame computer. We are given a set of electronic components, whose size is typically between 200 and 1000. The most important property of the electronic circuits – for our purposes – is the area that these components cover. The electronic components have to be integrated on printed circuit boards, multi chip modules or other devices. Each of these devices (whose number is usually between 5 and 10) is defined by several technical properties that we do not intend to describe here. Two properties of devices are important for us. Every device k has a *capacity* F_k, representing its "area" or the weight it can hold and a *cut capacity* S_k, describing the number of wires that can be connected to this device. The electronic components have certain contact points, called pins, from which wires can extend to pins of other components. In the logical design phase it is determined which pins of which components have to be connected by a wire to ensure certain functional properties. It is customary to call a collection of pins that have to be connected a *net*.

The task is to assign the electronic components to the devices in such a way that a certain objective function is minimized and a number of technical side constraints are satisfied. Among them are two essential requirements.

- For each device k the sum of the areas of the electronic components that are assigned to this device must not exceed the capacity F_k.
- The number of nets that must leave some device k must not exceed its cut capacity S_k.

The mathematical problem that arises by thoroughly modeling all (or at least the most important) aspects of this question is a rather complicated integer program. The full model appears to be hopelessly difficult – at least for the present state of integer programming technology. Thus, we investigated a hierarchy of combinatorial relaxations of the complete model. A first relaxation of the general model is the multiple knapsack problem, where we neglect the nets completely and concentrate on the packing aspect of the problem. In knapsack terminology, the items correspond to the electronic components and the devices to the knapsacks. An appropriate objective function for the multiple knapsack problem is determined heuristically. For more details we refer the reader to Ferreira, Grötschel, Kiefl, Krispenz, Martin, and Weismantel [1993].

The second application we have in mind arises in the layout of electronic circuits. Here, one major subproblem (the so-called placement problem) is to assign a set of logical units (cells) to locations on a given rectangle (silicon) subject to certain technical side constraints, see also the somewhat more detailed description on page 27. In general, cells are of rectangular shape and have a particular weight representing their area. Due to the inherent complexity of the placement problem and its large scale (typically up to several thousands of cells have to be assigned), it is further decomposed in practice. In a first step, the given rectangle is subdivided and it is determined which cells are assigned to which of the subareas such that the total weight of the cells that are assigned to the same subarea does not exceed the corresponding area capacity. This process of iteratively subdividing areas and assigning cells to subareas is continued until every subarea contains at most one cell. If we interpret cells as items and subareas as knapsacks, we can associate with every problem arising in this decomposition scheme a multiple knapsack problem. In fact, the multiple knapsack problem does not reflect the complete situation, since – besides the area requirements – there are many additional side constraints which are to be taken into account and a complicated objective function must be minimized which strongly depends on the underlying fabrication technology. Nevertheless, solving the corresponding multiple knapsack problems seems to be a reasonable starting point to attack the much more complicated placement problems.

The third application arises in the sugar cane alcohol production in Brazil. Since the late 70's, sugar cane alcohol has been used in Brazil as fuel for cars and plays an important role in Brazilian agriculture. Farmers sell their sugar cane to alcohol producers. The producers themselves collect the sugar cane from the farmers and deliver it to one of their factories. Each plantation i produces a certain amount f_i of sugar cane, and each factory k is able to treat a maximum amount F_k of sugar cane. The problem, from the side of the alcohol producer, is now to decide which plantation production to deliver to which factory such that the capacity constraints are satisfied and the transportation cost is minimized. The transportation cost of delivering the production of plantation i to factory k is proportional to the production of the plantation and the distance between the plantation and the factory. We obtained two real data instances of this problem. In both instances the plantations must be distributed to 4 factories. In the first instance there are 450 plantations while in the second one are 370.

In detail, the chapter is organized as follows. In Section 2.2 we report on some basic results for the multiple knapsack polytope and related polyhedra. Section 2.3 is devoted to the 0/1 knapsack polytope, i. e., the special case of the multiple knapsack problem where $|M| = 1$. In Section 2.4 we describe joint inequalities, inequalities that combine at least two knapsacks. Section 2.5 is devoted to separation algorithms for the described classes of inequalities and to our primal heuristics. Further implementation issues are also discussed in this section. We have tested our branch-and-cut algorithm on instances arising in the applications mentioned above. The computational results we obtained are shown in Section 2.6.

Other work on the multiple knapsack problem includes the following. The multiple knapsack problem and especially the single knapsack problem have been extensively treated in the literature from an algorithmic point of view, see the excellent book of Martello and Toth [1990]. However, the multiple knapsack polytope P_{MK} and generalizations of it have not been studied to the same extent. In a few papers we find investigations in this direction. Gottlieb and Rao [1990a], [1990b] study the *generalized assignment problem*, a generalization of the multiple knapsack problem, where every item i may have a particular weight f_{ik} for each knapsack k. They present some new classes of inequalities for the associated polytope P and show that the non-trivial facet-defining inequalities from the 0/1 knapsack polytope are facet-defining for P as well. Also related is the work by Crowder, Johnson, and Padberg [1983] who solve general 0/1 linear programs via a cutting plane algorithm, where they interpret each row of the constraint matrix as a knapsack problem and use single knapsack inequalities to strengthen the LP relaxation. This approach is very common to solve general integer programs, see Chapter 5 for more details.

2.2 The Multiple Knapsack Polytope: Some Observations

Let an instance $\mathrm{MK}_w\,(N, M, f, F, c)$ of the weighted multiple knapsack problem be given. For the study of the multiple knapsack polytope $P_{\mathrm{MK}}\,(N \times M, f, F)$ we may neglect the objective function c and denote from now on an instance of the multiple knapsack problem by $\mathrm{MK}\,(N, M, f, F)$, where the prefix "weighted" is omitted.

It will turn out that we often refer to subinstances of the problem where certain items are not feasible for certain knapsacks. Thus, we define the polyhedron in a more general framework. For $T \subseteq N \times M$, let $A := \{i \in N$: there exists a knapsack $k \in M$ with $(i,k) \in T\}$ and $B := \{k \in M$: there exists an item i such that $(i,k) \in T\}$. We define the *multiple knapsack polytope* by

$$(2.3)\qquad \begin{aligned} P_{\mathrm{MK}}\,(T, f, F) := \mathrm{conv}\{x \in \{0,1\}^T : \quad & \sum_{i:(i,k)\in T} f_i x_{ik} \le F_k, \quad k \in B, \\ & \sum_{k:(i,k)\in T} x_{ik} \le 1, \qquad i \in A\}. \end{aligned}$$

The polytope corresponding to the multiple knapsack problem defined in Section 2.1 coincides with $P_{\mathrm{MK}}\,(N \times M, f, F)$, which we often abbreviate by P_{MK}. It is easy to see that P_{MK} is full dimensional if and only if $f_i \le F_k$ for all $i \in N$ and $k \in M$. Similarly, the dimension of the polytope $P_{\mathrm{MK}}\,(T, f, F)$ equals $|T|$ if and only if $f_i \le F_k$ for all $(i,k) \in T$. In the following we assume that $f_i \le F_k$ for all $(i,k) \in T$. The *trivial inequalities* $x_{ik} \ge 0$ define facets of P_{MK} for all $i \in N, k \in M$. In case $m \ge 2$, the SOS constraints define facets of P_{MK} as well.

The multiple knapsack problem is a generalization of the (single) 0/1 *knapsack problem*, where $|M| = 1$. That is, we are given some knapsack of capacity $F \in \mathbb{N}$ and a set N of items, each having some weight $f_i \in \mathbb{N}$ and some profit $c_i \in \mathbb{R}$. The problem is to find a subset of items whose total weight fits into the knapsack and that maximizes the total profit. In analogy to the definitions of MK_w, MK, and P_{MK}, we denote by $\mathrm{K}\,(N, f, F)$ and $\mathrm{K}_w\,(N, f, F, c)$, respectively, an instance of the (weighted) knapsack problem, and by

$$(2.4)\qquad P_{\mathrm{K}}\,(N, f, F) := \mathrm{conv}\{x \in \{0,1\}^N : \sum_{i\in N} f_i x_i \le F\}$$

the *knapsack polytope*.

Obviously, $P_{\mathrm{K}}\,(N, f, F_k)$ yields a relaxation of $P_{\mathrm{MK}}\,(N \times M, f, F)$ for every $k \in M$, and hence the question arises whether knowledge about these individual knapsack polyhedra can be used to describe the corresponding multiple knapsack polytope. Indeed, the answer to this question is yes. All non-trivial facet-defining inequalities associated with the single knapsack polytopes are inherited by P_{MK} (cf. Gottlieb and Rao [1990a]). More precisely, let $V \subseteq N$ and $k \in M$ and suppose $a^T x \leq \alpha$ is a non-trivial facet-defining inequality of $P_{\mathrm{K}}(V, f, F_k)$, then $\overline{a}^T x \leq \alpha$ defines a facet of $P_{\mathrm{MK}}(V \times M, f, F)$, where $\overline{a} \in \mathbb{R}^{V \times M}$ with $a_{il} := a_i$ if $l = k$ and $a_{il} := 0$ otherwise, for $i \in V$.

We call a valid inequality $\overline{a}^T x \leq \alpha$ for $P_{\mathrm{MK}}(N \times M, f, F)$ *individual* if there is some $k \in M$ such that $\overline{a}_{il} = 0$ for all $l \in M \setminus \{k\}$. Otherwise, we call the inequality *joint*. In the following we deal with the single knapsack polytope $P_{\mathrm{K}}(N, f, F)$.

2.3 The 0/1 Knapsack Polytope

Consider an instance $\mathrm{K}(N, f, F)$ of the knapsack problem, i. e., a set $N = \{1, \ldots, n\}$ of items with weights $f_i, i \in N$, and a capacity F, and the polytope $P_{\mathrm{K}}(N, f, F)$ as defined in (2.4). In this section we summarize results known for 0/1 knapsack polytope $P_{\mathrm{K}}(N, f, F)$.

The knapsack polytope $P_{\mathrm{K}}(N, f, F)$ is full dimensional if and only if $f_i \leq F$ for all $i \in N$, which we assume in the following. If there exists a subset $S \subseteq N$ and an element $s \in S$ such that $\sum_{i \in S} f_i > F$ and $\sum_{i \in S \setminus \{s\}} f_i < F$, then the polyhedron $\{x \in [0,1]^N : f^T x \leq F\}$ is not integral. If $f_i = 1$ for all $i \in N$, then the polyhedron $\{x \in [0,1]^N : f^T x \leq F\}$ is integral. The trivial inequalities $x_i \geq 0$ define facets of the knapsack polytope. Every facet-defining inequality $a^T x \leq \alpha$ that is not a positive multiple of the trivial inequalities $x_i \geq 0$ satisfies $a \geq 0$ and $\alpha > 0$.

A set $S \subseteq N$ is a *cover* if its weight exceeds the capacity, i. e., if $\sum_{i \in S} f_i > F$. With the cover S one can associate the *cover inequality*

$$\sum_{i \in S} x_i \leq |S| - 1$$

that is valid for the knapsack polyhedron $P_{\mathrm{K}}(N, f, F)$. If the cover is *minimal*, i. e., if $\sum_{i \in S \setminus \{s\}} f_i \leq F$ for all $s \in S$, the inequality is called *minimal cover inequality (with respect to S)*.

In Balas [1975], Padberg [1975], Hammer, Johnson, and Peled [1975], and Wolsey [1975] it was shown that the minimal cover inequality defines a facet of $P_{\mathrm{K}}(S, f, F)$. Laurent and Sassano [1992] showed that n minimal cover inequalities suffice to describe the knapsack polytope if the vector $f = (f_1, \ldots, f_n)$ is weakly superdecreasing, i. e., if $\sum_{i \geq q} f_i \leq f_{q-1}$, for all $q = 2, \ldots, n$

Another well-known class of individual inequalities are $(1,k)$-configuration inequalities that were introduced by Padberg [1980]. A *$(1,k)$-configuration* consists of an independent set S, i. e., a set S such that $\sum_{i \in S} f_i \leq F$, plus one additional item z such that every subset of S of cardinality k, together with z, forms a minimal cover. A $(1,k)$-configuration $S \cup \{z\}$ gives rise to the inequality

$$\sum_{i \in S} x_i + (|S| - k + 1) x_z \leq |S|,$$

which is called *$(1,k)$-configuration inequality (with respect to $S \cup \{z\}$)*. Note that a minimal cover S is a $(1, |S|-1)$-configuration, and vice versa, a $(1,k)$-configuration inequality (with respect to $S \cup \{z\}$) that satisfies $k = |S|$ is a minimal cover. In Padberg [1980] it was shown that the $(1,k)$-configuration inequality defines a facet of $P_{\mathrm{K}}(S \cup \{z\}, f, F)$. Padberg also proved that if the set N of items defines a $(1,k)$-configuration, then the convex hull of the associated knapsack polyhedron is given by the trivial inequalities and the set of all $(1,l)$-configuration inequalities defined by $T \subseteq N \setminus \{z\}$, where $T \cup \{z\}$ is a $(1,l)$-configuration for some $l \leq k$.

Inequalities derived from both covers and $(1,k)$-configurations are special cases of *extended weight inequalities* that have been introduced by Weismantel [1997].

Consider a subset $T \subseteq N$ with $f(T) < F$ and let $r := F - f(T)$. The inequality

$$\sum_{i \in T} f_i x_i + \sum_{i \in N \setminus T} (f_i - r)^+ x_i \leq f(T). \tag{2.5}$$

is called *weight inequality with respect to* T. The name *weight inequality* reflects that the coefficients of the items in T equal their original weights and the number $r := F - f(T)$ corresponds to the residual capacity of the knapsack when $x_i = 1$ for all $i \in T$.

Weight inequalities are needed to describe the knapsack polyhedron when the weights of all items are equal to 1 or greater than $F/2$. Indeed in this case, the weight inequalities together with the trivial inequalities, the original knapsack inequality and the (lifted) minimal cover inequality $\sum_{i \in N : f_i \geq \lfloor \frac{F}{2} \rfloor + 1} x_i \leq 1$ describe the knapsack polytope completely.

There is a natural way to extend weight inequalities by (i) replacing the original weights of the items by relative weights and (ii) resorting to the method of sequential lifting, see Appendix C. To introduce this extension we need the following definition.

Let $T \subseteq N$ such that $f(T) \leq F$. For every $v \in \mathbb{N}$, the *covering number of* v *associated with* T denoted by $\phi(v)$ and defined as

$$\phi(v) := \min\{|S| : S \subseteq T, f(S) \geq v\}. \tag{2.6}$$

The covering number of the sum of two numbers plus one is an upper bound for the sum of the covering number of the single numbers, i. e., for $v, w \in \mathbb{N}$ with $v + w \leq f(T)$, the relation $\phi(v+w) + 1 \geq \phi(v) + \phi(w)$ holds.

We will prove this statement in a more general setting in Lemma 8.2.3. The relation will be used later to bound the coefficients of the extended weight inequality, which is defined as follows.

Let $T \subseteq N$, $f(T) \leq F$, define $r := F - f(T)$ and denote by S the subset of $N \setminus T$ such that $f_i \geq r$ for all $i \in S$. The *(uniform) extended weight inequality* associated with T and the permutation $\pi_1, \ldots \pi_{|S|}$ of the set S is of the form

$$\sum_{i \in T} x_i + \sum_{i \in S} w_i x_i \leq |T|, \tag{2.7}$$

where $w_{\pi_1} = \phi(f_{\pi_1} - r)$, and, for $i = 2, \ldots, |S|$, w_{π_i} is the exact lifting coefficient obtained by applying Algorithm C.2 from Appendix C, i. e., $w_{\pi_i} = |T| - \max\{\sum_{j \in T \cup \{\pi_1, \ldots, \pi_{i-1}\}} w_j z_j : z \in P_{\mathrm{K}}(T \cup \{\pi_1, \ldots, \pi_{i-1}\}, f_{T \cup \{\pi_1, \ldots, \pi_{i-1}\}}, F - f_{\pi_i})$.

Extended weight inequalities subsume the family of minimal cover and $(1,k)$-configuration inequalities, since for some $(1,k)$-configuration $S \cup \{z\}$ we have $w_z = \phi(f_z - r) = |S| - k + 1$. Extended weight as well as weight inequalities are valid for the knapsack polytope, see Weismantel [1997]. As the definition of an extended weight inequality requires the solution of an integer program for each item not in the starting set T, it is important to have bounds on the lifting coefficients. Weismantel showed that

$$\phi(f_i - r) - 1 \leq w_i \leq \phi(f_i - r), \tag{2.8}$$

which also follows from Proposition 8.2.2 (b) and Theorem 8.2.4. The algorithmic use of this relation is that one can derive sufficient and easy to verify conditions under which the lifting coefficient w_i can be set to the upper bound $\phi(f_i - r)$. We will exploit such a condition in the development of our separation algorithm for extended weight inequalities that we discuss in Section 5.3. Moreover, Weismantel

[1997] generalizes uniform extended weight inequalities to inequalities with arbitrary weights in the starting set T. For further references on the 0/1 knapsack polytope, see Aardal and Weismantel [1997].

2.4 Joint Inequalities

Besides individual inequalities there are classes of joint inequalities that are valid for the multiple knapsack polytope. In this section we present some of the joint inequalities we have found for the multiple knapsack polytope. These inequalities will be used in our cutting plane algorithm for solving practical problem instances. We refer the reader interested in further (facet-defining) inequalities to Ferreira, Martin, and Weismantel [1996] and Ferreira [1994]. In addition to the notation in Appendix A we denote by e_{ik} the unit vector in the vector space $\mathbb{R}^{N\times M}$ and define, for a vector $x' \in \{0,1\}^{N\times M}$ and some knapsack $k \in M$, $B_k(x') := \{i \in N : x'_{ik} = 1\}$.

2.4.1 The Multiple Cover Inequality

In Wolsey [1990] it was observed that, given sets $S \subseteq N$ and $J \subseteq M$ with $f(S) > F(J)$, the inequality

$$\sum_{j\in J}\sum_{i\in S} x_{ij} \leq |S| - 1$$

is valid for the polytope P_{MK}. If $|J| \geq 2$ a set of items S with the property $f(S) > F(J)$ is called a *multiple cover* (with respect to J) and the corresponding inequality *multiple cover inequality.* A set of items S is called a *minimal multiple cover* (with respect to J), if $f(S) > F(J)$ and, for all $s \in S$, there exists a valid assignment of all items in $S \setminus \{s\}$ to the knapsacks in J.

One can easily convince oneself that the multiple cover inequality does not always define a facet of P_{MK}. Thus, the question arises what are (necessary and sufficient) conditions under which the multiple cover inequality defines a facet. We have investigated this question and it turns out that, in general, these conditions are rather complicated and involve many (probably) unavoidable technicalities. We do not present the details here. However, the conditions simplify if we consider the special case of the multiple knapsack problem where the knapsack capacity values are all equal.

Theorem 2.4.1 *Let $MK(N,M,f,F)$ be an instance of the multiple knapsack problem where $F_k = F_l$ for all $k,l \in M$. Let $S \subseteq N$ be a minimal multiple cover for some $J \subseteq M$. Then, the multiple cover inequality*

$$\sum_{i\in S}\sum_{j\in J} x_{ij} \leq |S| - 1$$

defines a facet of $P_{MK}(S \times J, f, F)$ if and only if there exists an item $i \in S$ and a valid assignment x' of the items in $S\setminus\{i\}$ to the knapsacks in J such that $|B_k(x')| \neq |B_l(x')|$ for some $k,l \in J, k \neq l$.

Proof. We first prove that the condition is sufficient. Set $a := \sum_{i\in S}\sum_{j\in J} e_{ij}$ and $\alpha := |S| - 1$. The inequality $a^T x \leq \alpha$ is clearly valid. Let us prove that it defines a facet of $P_{\mathrm{MK}}(S \times J, f, F)$. Suppose that $b^T x \leq \beta$ defines a facet of $P_{\mathrm{MK}}(S \times J, f, F)$ such that $\mathrm{EQ}\,(a^T x \leq \alpha) \subseteq \mathrm{EQ}\,(b^T x \leq \beta)$. Let i_0 be an index such that $f_{i_0} = \min\{f_i : i \in S\}$ and let x^1 denote a valid assignment of the items in $S \setminus \{i_0\}$ to the knapsacks in J. Obviously, x^1 is in EQ $(a^T x \leq \alpha)$. Also, notice that for all $k \in J$ and $i \in B_k(x^1)$, the vector $x^1 - e_{ik} + e_{i_0k}$ is an element of EQ $(a^T x \leq \alpha)$. Thus, $b^T x^1 = b^T(x^1 - e_{ik} + e_{i_0k})$, yielding $b_{ik} = b_{i_0k}$. Moreover, since the capacities of the knapsacks are all equal, we can exchange the items of every pair of knapsacks and repeat the same arguments as above. Summing up, we conclude that, for every $k \in J$, there exists a constant c_k such that $b_{ik} = c_k$ for all $i \in S$.

In order to prove that $c_k = c_l$ for $k \neq l, k, l \in J$, let $i \in S$ be an item and let x' denote a valid assignment of the items in $S \setminus \{i\}$ to the knapsacks in J such that $|B_k(x')| \neq |B_l(x')|$ for some $k, l \in J, k \neq l$ as required in the condition. Since all knapsacks have the same capacity, we can construct a valid assignment $x'' = (x''_{ij})$ via:

$$x''_{ij} := \begin{cases} x'_{il}, & \text{for all } i \in S, j = k, \\ x'_{ik}, & \text{for all } i \in S, j = l, \\ x'_{ij}, & \text{otherwise.} \end{cases}$$

Clearly, x' and x'' belong to the face EQ $(a^T x \leq \alpha)$. Thus, $b^T x' = b^T x''$, yielding

$$|B_k(x')|c_k + |B_l(x')|c_l = |B_k(x')|c_l + |B_l(x')|c_k.$$

This implies $c_k = c_l$. Due to the uniform knapsack capacities, we can apply this construction for all other knapsacks and, finally obtain that $b^T x \leq \beta$ and $a^T x \leq \alpha$ are equal up to multiplication with a scalar, which completes the first part of the proof.

It remains to be shown that the condition is necessary. Suppose it is not satisfied, i.e., for all $x \in$ EQ $(a^T x \leq \alpha)$ and $k, l \in J, k \neq l$, $|B_k(x)| = |B_l(x)|$ holds. In this case, all $x \in$ EQ $(a^T x \leq \alpha)$ satisfy the equation $\sum_{i \in S} x_{ik} = \frac{|S|-1}{|J|}$, for all $k \in J$. Thus, the inequality cannot be facet-defining. □

2.4.2 Extension of Facet-defining Inequalities

In this subsection we present a general procedure that allows the extension of particular classes of inequalities. As we will see this procedure can be iteratively applied starting, for instance, from a minimal or multiple cover inequality. This way we obtain many new valid and facet-defining inequalities.

Theorem 2.4.2 *Let an instance $MK(N, M, f, F)$ of the multiple knapsack problem be given, sets $A \subseteq N$, $B \subseteq M$ and an inequality $a^T y \leq \alpha$ that is facet-defining for $P_{MK}(A \times B, f, F)$ and that satisfies the following additional requirement:*

($\star$) *For all $\tilde{A} \subseteq A$ with $|\tilde{A}| \geq 2$ the following holds: every assignment $y \in P_{MK}(A \times B, f, F)$ with $y_{ik} = 0$ for all $i \in \tilde{A}$, $k \in B$, satisfies $a^T y \leq \alpha - |\tilde{A}| + 1$.*

We choose a positive integer $r \leq \min\{|N \setminus A|, |M \setminus B|\}$, non-empty sets $T_1, \ldots, T_r$ that are mutually disjoint subsets of $N \setminus A$, and a subset $\{k_1, \ldots, k_r\}$ of $M \setminus B$. Let us further define $\beta_v := \max\{|G| : G \subseteq T_v, f(G) \leq F_{k_v}\}, v = 1, \ldots, r$. Then, the inequality

$$a^T x + \sum_{v=1}^{r} \sum_{i \in A \cup T_v} x_{ik_v} \leq \alpha + \sum_{v=1}^{r} \beta_v \tag{2.9}$$

is valid for the polytope $P_{MK}(A \times B \cup \bigcup_{v=1}^{r}((T_v \cup A) \times \{k_v\}), f, F)$ if and only if $\tilde{T}_v \cup \{i\}$ is a cover with respect to k_v for all $i \in A$, $\tilde{T}_v \subseteq T_v$, $|\tilde{T}_v| = \beta_v$ and $v = 1, \ldots, r$.

Proof. For the ease of notation let us refer to inequality (2.9) by $b^T x \leq \beta$ and set $Q := A \times B$.

We first prove that the condition is necessary. Suppose, there exists an index $v \in \{1, \dots, r\}$, an item $i_0 \in A$ and a set $\tilde{T}_v \subseteq T_v$, $|\tilde{T}_v| = \beta_v$ such that $\tilde{T}_v \cup \{i_0\}$ is not a cover with respect to k_v. Let $\tilde{T}_w \subseteq T_w, w \in \{1, \dots, r\} \setminus \{v\}$, $|\tilde{T}_w| = \beta_w$, $f(\tilde{T}_w) \leq F_{k_w}$.

There exists an assignment $y' \in \mathrm{EQ}\,(P_{\mathrm{MK}}\,(Q, f, F), a^T y \leq \alpha)$ with $y'_{i_0 k} = 0$ for all $k \in B$, because $a^T y \leq \alpha$ defines a facet. Set $x' = (x'_{ik})$ via:

$$x'_{ik} := \begin{cases} y'_{ik}, & \text{for all } (i,k) \in Q, \\ 1, & \text{for all } i \in \tilde{T}_w, k = k_w, w \in \{1, \dots, r\} \setminus \{v\}, \\ 1, & \text{for all } i \in \tilde{T}_v, k = k_v, \\ 0, & \text{otherwise.} \end{cases}$$

Since $\tilde{T}_v \cup \{i_0\}$ does not define a cover with respect to knapsack k_v, $x' + e_{i_0 k_v}$ is a valid assignment yielding $b^T(x' + e_{i_0 k_v}) = a^T y' + \sum_{w \in \{1, \dots, r\} \setminus \{v\}} \beta_w + \beta_v + 1 > \alpha + \sum_{v=1}^r \beta_v$. This implies that the condition is necessary, indeed.

In order to prove the converse direction, let us assume that the inequality is not valid for the polytope $P_{\mathrm{MK}}\,(Q \cup \bigcup_{v=1}^r (T_v \cup A) \times \ \{k_v\}, f, F)$, i.e., there exists an assignment $x \in P_{\mathrm{MK}}\,(Q \cup \bigcup_{v=1}^r ((T_v \cup A) \times \{k_v\}), f, F)$ with $b^T x > \beta$. Set $\tilde{T}_v := \{i \in T_v : x_{ik_v} = 1\}$ and $A_v := \{i \in A : x_{ik_v} = 1\}, v = 1, \dots, r$. Since the inequality $a^T y \leq \alpha$ holds for all $y \in P_{\mathrm{MK}}\,(Q, f, F)$, there exists some $v \in \{1, \dots, r\}$ satisfying $|A_v \cup \tilde{T}_v| > \beta_v$. Let $V \subseteq \{1, \dots, r\}$ denote the subset of knapsacks with $|A_v \cup \tilde{T}_v| > \beta_v$ for all $v \in V$. Due to the condition, every subset of T_v of cardinality β_v and one element from A defines a cover with respect to k_v $(v \in V)$. This implies that $|\tilde{T}_v| \leq \beta_v - 1$, thus forcing A_v to be greater or equal than two, which holds for all $v \in V$. Moreover, $A_v \cap A_w = \emptyset$ for all $v, w \in V, v \neq w$. Summing up, due to requirement $(\star)$ we obtain

$$\begin{aligned} b^T x &\leq \alpha - \sum_{v \in V} |A_v| + 1 + \sum_{v \in \{1, \dots, r\} \setminus V} \beta_v + \sum_{v \in V} (|\tilde{T}_v| + |A_v|) \\ &\leq \alpha - \sum_{v \in V} |A_v| + 1 + \sum_{v \in \{1, \dots, r\} \setminus V} \beta_v + \sum_{v \in V} (\beta_v + |A_v|) - |V| \\ &= \beta + 1 - |V| \leq \beta. \end{aligned}$$

This contradicts the assumption that x is a point violating the inequality. Thus, the inequality is valid, which completes the proof. □

If we strengthen the requirements and conditions of Theorem 2.4.2, the extended inequality will also be facet-defining. The subsequent theorem (which we give without a proof) states necessary and sufficient conditions for the extended inequality to be facet-defining.

Theorem 2.4.3 *We assume the same assumptions as in Theorem 2.4.2. In addition, we require that $T_v \cup \{i\}$ is a minimal cover with respect to knapsack k_v for all $i \in A$ and $v = 1, \dots, r$. Then, the inequality*

$$a^T x + \sum_{v=1}^{r} \sum_{i \in A \cup T_v} x_{ik_v} \leq \alpha + \sum_{v=1}^{r} |T_v| \tag{2.4.3}$$

defines a facet for $P_{MK}\,(A \times B \cup \bigcup_{v=1}^r (T_v \cup A) \times \{k_v\}, f, F)$ if and only if, for every $v \in \{1, \dots, r\}$, there exist a subset $\tilde{A} \subseteq A$ with $|\tilde{A}| \geq 2$, an item $t_v \in T_v$, and an assignment $y \in P_{MK}\,(A \times B, f, F)$ such that $a^T y = \alpha - |\tilde{A}| + 1$, $y_{ik} = 0$ for all $i \in \tilde{A}$, $k \in B$ and $f(T_v \setminus \{t_v\}) + f(\tilde{A}) \leq F_{k_v}$.

The two theorems can easily be generalized by using instead of $A \times B$ any arbitrary "starting sets" $T \subseteq N \times M$. At first sight, the requirement ($\star$) seems to be quite restrictive, but it turns out that many inequalities satisfy this condition. For example, the minimal cover and multiple cover inequality are such inequalities. Moreover, when we apply Theorem 2.4.3 to the minimal cover or multiple cover inequality, respectively, the resulting extended inequality still satisfies requirement ($*$). Thus, a repeated extension in the "spirit" of Theorem 2.4.3 is possible. Moreover, one can convince oneself that, for example, the r-fold repetition of this "extension procedure" (i. e., at each time we extend the original inequality by one set of items T_v and one knapsack k_v) leads to an inequality which is different from the one obtained by a simultaneous extension with sets $T_1, \ldots, T_r$ and knapsacks $k_1, \ldots, k_r$ is applied to the original inequality.

2.5 Algorithmic Aspects

In this section we briefly discuss the separation algorithms for the classes of inequalities presented in Sections 2.2 and 2.4 and present our primal heuristics. We also point to features that supplement the general description given in Appendix B in order to solve problems of practical size.

Let us first note that for the applications we have in mind all items must be assigned to the knapsacks, i. e., the SOS constraints must be satisfied with equality. Thus, instead of using the SOS constraints we actually work with the corresponding equalities $\sum_{k \in M} x_{ik} = 1$ $(i \in N)$. From now on, we will call a solution *feasible* for the multiple knapsack problem only if all items are assigned to the knapsacks, i. e., if all SOS constraints are satisfied with equality. In addition, in all applications the problems are formulated as minimization problems unlike the integer programming formulation in (2.1), and we treated them as such.

We also point out that our initial linear program in the algorithm is composed of the knapsack inequalities, the SOS inequalities and the trivial inequalities $0 \leq x_{ik} \leq 1$, for all $i \in N$, $k \in M$. Thus, we may assume in the following that the actual LP solution satisfies these constraints.

2.5.1 Separation Algorithms

All the classes of inequalities shown in the previous section are generalizations of cover inequalities. Obviously, the problem of finding a minimum (with respect to some weighting of the items) cover is $\mathcal{NP}$-hard, because it reduces to the single knapsack problem. It turns out that also the separation problem for the minimal cover inequality reduces to the single knapsack problem, implying that it is $\mathcal{NP}$-hard, see Ferreira [1994] or Klabjan, Nemhauser, and Tovey [1996]. Though these results do not prove that all the other separation problems are $\mathcal{NP}$-hard as well, we strongly conjecture that they are. Thus, we put our emphasis on developing separation heuristics, which we will discuss in the following.

Among the individual inequalities we separate *minimal cover* and $(1, k)$-*configuration inequalities*. We implemented various greedy-type heuristics for these two classes, see Ferreira, Martin, and Weismantel [1996]. We will come across some of the issues of these separation procedures in Section 5.3 where we outline the separation of extended weight inequalities. Here we focus on the separation of joint inequalities.

Our procedure for separating *multiple cover inequalities* is a two-stage process. First, we determine a set $\mathcal{M}$ of candidate sets $M' \subseteq M$. For each of these sets $M' \in \mathcal{M}$ we then try to determine a set of items that defines a multiple cover. The latter problem is solved by applying our minimal cover separation routine to the "aggregated" knapsack with capacity $F(M')$, see also Borndörfer and Weismantel [1997a]. More precisely, suppose, $\bar{x} \in \mathbb{R}^{N \times M}$ is the current fractional point and $M' \subseteq M$ is the set of knapsacks for which we want to find a multiple cover. We define the instance $\mathrm{K}(N, f, C)$ of the single knapsack problem by setting $C := F(M')$. Moreover we define a new fractional point, $y \in \mathbb{R}^N$ say, where $y_i := \sum_{k \in M'} \bar{x}_{ik}$ for all $i \in N$. With input $\mathrm{K}(N, f, C)$ and y, the routines for finding minimal cover inequalities are called. If these calls yield a violated minimal cover inequality, it is transformed back to the original space and, hence, corresponds to a violated multiple cover inequality.

Unfortunately, there are $2^{|M|} - (|M| + 2)$ different sets of knapsacks for which we could perform the algorithm just described. This would, even for small $|M|$, result in an immense running time. Therefore, we generate a set $\mathcal{M}$ of candidates $M' \subseteq M$ according to some heuristic rules which we briefly explain now. For every item $i \in N$ we determine $B_i := \{k \in M : \bar{x}_{ik} > 0\}$, where $\bar{x}$ is the current LP solution. If $2 \leq |B_i| \leq |M| - 1$, we consider B_i as one candidate set for which we try to find a multiple cover as described before. In case $B_i = M$, we simply add all subsets of M of cardinality $|M| - 1$ to the set $\mathcal{M}$ of candidate sets. The idea for this (heuristic) selection of the candidate sets is that B_i is a subset of knapsacks where still (at least) one item is in conflict with its "correct position". Thus, additional inequalities are needed that provide further information on how to handle this conflict. Proceeding this way, the number of different candidate sets that are generated does not exceed the number $|N| + |M| - 1$.

The final separation procedure we are using in our implementation deals with a special class of inequalities as described in Theorem 2.4.2. Let S be a cover with respect to some knapsack k, let $l \in M \setminus \{k\}$ and $T \subseteq N \setminus S$. Obviously, the minimal cover inequality $\sum_{i \in S} x_{ik} \leq |S| - 1$ satisfies property $(\star)$ of Theorem 2.4.2. Thus, due to Theorem 2.4.2 the inequality

$$\sum_{i \in S} (x_{ik} + x_{il}) + \sum_{i \in T} x_{il} \leq |S| + |T| - 1 \tag{2.10}$$

is valid for the polytope P_{MK} if and only if $T \cup \{i\}$ is a cover with respect to l for all $i \in S$. We call this inequality an *extended cover inequality.* We have implemented a heuristic procedure for finding extended cover inequalities along the following lines.

Algorithm 2.5.1 *Separation of Extended Cover Inequalities.*

(1) Choose two knapsacks $k, l \in M$, $k \neq l$ and solve the linear program:

$$\begin{aligned} \min \quad & \textstyle\sum_{i \in N} (1.0 - \bar{x}_{ik} - \bar{x}_{il}) s_i \\ & \textstyle\sum_{i \in N} f_i s_i > F_k \\ & 0 \leq s_i \leq 1 \textit{ for all } i \in N. \end{aligned}$$

(2) Set $S := \{i \in N : s_i > 0\}$.

(3) Reduce S to a minimal cover.

(4) Let s_{min} be an item in S with minimum weight.

(5) Solve the linear program:

$$\begin{aligned} \min \quad & \textstyle\sum_{i \in N \setminus S} (1.0 - \bar{x}_{il}) t_i \\ & \textstyle\sum_{i \in N \setminus S} f_i t_i > F_l - f_{s_{min}} \\ & 0 \leq t_i \leq 1 \textit{ for all } i \in N \setminus S. \end{aligned}$$

(6) Set $T := \{i \in N \setminus S : t_i > 0\}$.

(7) Reduce T so that $T \cup \{s_{min}\}$ is a minimal cover.

(8) Lift the extended cover inequality with respect to S, T, k and l (for lifting *see Appendix C).*

The success of the discussed separation algorithms is documented in Section 2.6.

2.5.2 Primal Heuristics

We have implemented several LP-based heuristics in order to obtain good upper bounds on the value of the optimal solution. These heuristics are based on rounding the current linear programming solution $\bar{x}$.

In the first heuristic we proceed as follows. We set $F'_k := F_k$ for all $k \in M$ and determine the item i whose value $\max\{\bar{x}_{ik} : k \in M \text{ and } f_i \leq F'_k\}$ is maximum. Let i^* be the "maximum" item and k^* the corresponding knapsack. We assign i^* to knapsack k^* and update the value F'_{k^*} by setting $F'_{k^*} := F'_{k^*} - f_{i^*}$. We continue this way until a feasible solution is found (i. e., all items are assigned) or no further item can be assigned.

The second heuristic differs from the first one by selecting the item that is assigned next randomly. More precisely, we determine a random sequence of the items, and, according to this sequence, we compute $\max\{\bar{x}_{ik} : k \in M \text{ and } f_i \leq F'_k\}$. If there exists no more feasible knapsack for item i we stop. Otherwise, we assign item i to a knapsack k^* where the maximum value is attained, update F'_{k^*} and continue.

In the third heuristic, we interpret each vector $(\bar{x}_{ik})_{k \in M}$, for $i \in N$, as a probability distribution, i. e., we assign item i to knapsack k with probability $\bar{x}_{ik}$. The sequence according to which the items are chosen is again randomly determined. This procedure is performed several times depending on a parameter that is a multiple of the number of fractional variables.

It turns out that none of these heuristics is superior to the others. In most examples, the first two procedures are more successful at the very beginning, whereas the third one performs better in the sequel.

Moreover, we have implemented an improvement heuristic that is based on the ideas of Fiduccia and Mattheyses [1982]. The procedure runs in a number of passes. Each pass starts with the currently best feasible solution. Now we try to change the solution by moving an item from one knapsack to another such that the new solution is still feasible. Among all feasible moves we choose the best one and fix the corresponding item for the current pass. Note that we also perform the move if it leads to a worse solution. This way it might be possible to get out of locally optimal solutions. The pass ends if all items have been moved or if there is no feasible move for the not yet considered items. Among these at most $|N|$ new solutions we choose the best one and update the globally best solution.

It turns out that these three primal heuristics together with the improvement heuristic perform quite well and even find the optimal solution in many cases, see next section.

2.5.3 Further Issues

In Appendix B we have already discussed the general outline of a branch-and-cut algorithm. Our algorithm basically follows this scheme. Here we want to stress just one aspect that was important for the solution of practical problem instances.

We found that fixing variables by the reduced cost criterion is very effective for some of the problem instances (for an explanation of reduced cost fixing, see Appendix B). Table 2.1 shows for some examples the number of variables (in percentages) that are fixed by this procedure. For a description of the problems including the reduction parameter in Column *Red.* see the next section. As we see, it is possible to fix up to 90% of the variables.

Problem	Red.	Variables fixed	
cl2	0.0%	2565	(23.54%)
cl2	4.0%	2135	(19.59%)
dm1	36.8%	89	(8.79%)
dm2	30.0%	1256	(27.12%)
sa1	0.0%	1653	(91.83%)
sa2	0.0%	1072	(72.43%)

Table 2.1: Reduced cost fixings for some multiple knapsack instances.

2.6 Computational Results

In this section we report on computational experiences with our cutting plane based algorithm. We have tested the algorithm on multiple knapsack problem instances arising in the design of main frame computers, in the layout of electronic circuits and in the production of sugar cane alcohol.

Before presenting the results in detail, let us make some general comments on our experiences with the problem instances. As described in Section 2.5, we have separated cover, $(1, k)$-configuration, multiple cover and extended cover inequalities. The number of cover and $(1, k)$-configuration inequalities we found are approximately the same for all examples. For joint inequalities, however, the situation is different. Almost all violated joint inequalities are multiple covers. In fact, for only one example were extended covers found. We do not know whether the extended cover inequalities are indeed rare in these particular problem instances, or whether the performance of our separation heuristic is too poor. In the following tables we will just distinguish between individual and joint inequalities.

Let us first focus on the class of problems that arise in the *design of main frame computers*. We obtained the problem instances from Siemens-Nixdorf, Munich. Table 2.2 summarizes the data. Instances coming from this application are abbreviated by dm in the tables. Column 2 and 3 give the number of items and knapsacks, respectively, whereas in Columns 4 and 5 the total weight of the items and the total sum of the capacities are shown.

Problem	$\|N\|$	$\|M\|$	$\sum_{i \in N} f_i$	$\sum_{k \in M} F_k$
dm1	257	4	83827	132704
dm2	772	6	284608	423972

Table 2.2: Design of main frame computers: Problem data

Table 2.3 presents the solutions obtained by our algorithm. Neither individual nor joint inequalities where necessary to obtain the objective function value of an optimal solution, which was already found by the primal heuristics in the first iteration. The CPU Times are in seconds obtained on a Sun SPARC 4/50.

Problem	Opt. Sol.	Ind. Ineq.	Joint Ineq.	CPU Time
dm1	236250	0	0	1.62
dm2	81120	0	0	5.92

Table 2.3: Results for problems from the design of mainframe computers

The fact that both problem instances are trivial is not surprising, since the total sum of the knapsack capacities is much bigger than the sum of the weights of the items. The reason for that is that after assigning the items to the devices the nets must be connected by wires which requires a certain amount of space. The real amount of space that is necessary for connecting the wires is not available in advance. Thus, the numbers in Column 5 of Table 2.2 are only a rough estimate. A usual procedure in practice is to start with some initial capacities of the devices and try to find a solution that assigns the modules to the devices and connects the nets by wires. If this succeeds, the capacities of the devices are reduced, the whole problem is solved again, and it is continued in this way until no further area reduction is possible. In fact, one of the main goals in the design of main frame computers is to reduce the available amount of space as far as possible. So, from a practical point of view a very interesting question is how far the capacities of the devices can be reduced at most. We followed this question and iteratively reduced the total amount of the knapsack capacities.

Problem	Red.	Opt. Sol.	Ind. Ineq.	Joint Ineq.	B&B	CPU
dm1	36.75%	236250	18	15	1	8.78
dm1	36.8%	236250	18	15	1	8.87
dm2	27%	81134	9	16	1	40.75
dm2	28%	81176	42	41	1	5:22.25
dm2	29%	81204	25	29	1	1:08.30
dm2	30%	81302	33	15	1	36:41.92
dm2	31%	81486	2547	3308	105	485:22.03
dm2	32%	81736	44	62	5	56:35.20

Table 2.4: Results for reduced examples from the design of main frame computer

Table 2.4 summarizes our results. A reduction of 36.85% in example dm1 and a reduction of 33% in example dm2 leads to infeasibility, since the total available knapsack capacity is less than the sum of the weights of the items. Column 2 shows the amount of reduction, Columns 3 to 6 present the value of the optimal solution, the number of individual and joint inequalities found by our separation algorithms, the number of branch-and-bound nodes, and the total CPU Time (min:sec). The percentage of separation time is between 20% and 50% for these examples. All, except the last two, problem instances in Table 2.4 are solved to optimality without branching.

Let us now turn to the examples arising in the *layout of electronic circuits*. Table 2.5 summarizes the data. Here, the instances are abbreviated by the symbol cl in the tables. The ratio between the total weight of the items and the total available knapsack capacity is much closer to one than it is the case for the instances in Table 2.2. One might expect that these instances are more difficult than the original dm examples.

However, the results are very similar. The first lower bound provides the value of the optimal solution in almost all examples. A possible explanation for this fact is that in these examples the weights of the items are similar and that there are many items with small weights and identical objective function value for all knapsacks. In all cases for which the value of the first LP is already equal to the value of the optimal solution our primal heuristics find an optimal solution in the first iteration of the algorithm. The only non-trivial example is cl2, where indeed individual and joint inequalities were necessary to find the optimal solution. In Table 2.6 we show the value of an optimal solution, the number of individual and joint inequalities found by our separation algorithms and the total CPU Time (min:sec). Here, about 50%

Problem	$\|N\|$	$\|M\|$	$\sum_{i \in N} f_i$	$\sum_{k \in M} F_k$
cl1	2292	16	9522	10000
cl2	681	16	2571	2704
cl3	2669	16	6762	7104
cl4	1021	16	4031	4240
cl5	68	16	260	288
cl6	6112	16	25392	26672

Table 2.5: Layout of electronic circuits: Problem data

of the CPU Time was spent in the separation routines.

Problem	Opt. Solution	Ind. Ineq.	Joint Ineq.	CPU Time
cl1	2292	0	0	3:58.53
cl2	939.99	143	46	11:40.97
cl3	2669	0	0	3:26.42
cl4	1021	0	0	28.43
cl5	472	0	0	1.73
cl6	6112	0	0	25:28.62

Table 2.6: Results for problems arising in the layout of electronic circuits

Again, as in the design of main frame computers it is an interesting problem to determine the minimum area for which the problems still have a feasible solution. We created some problem instances by reducing the total amount of the capacities for the knapsacks until the total weight of the items exceeds the total available knapsack capacity. Even here, we solve all reduced examples, except the reduced instances of cl2, in the first iteration, a very astonishing fact. The reduced instances of cl2 are much more difficult. In all but one example the first lower bound does not give the optimal objective function value. To solve these problems to optimality without branching not only individual inequalities but also joint inequalities were necessary. Table 2.7 presents the results. Note that a reduction of 5% leads to infeasibility, since in this case the total sum of the knapsack capacities is less than the total weight of the items.

Problem	Red.	Opt. Sol.	Ind. Ineq.	Joint Ineq.	CPU Time
cl2	1%	946.99	164	94	11:18.48
cl2	2%	946.99	88	44	8:26.70
cl2	3%	960.99	163	119	14:33.45
cl2	4%	967.99	204	157	19:12.63

Table 2.7: Results for different reductions of problem example cl2

Our third set of instances comes from the application in *sugar cane alcohol production*. In Table 2.8 we show a description of the instances (they are abbreviated by sa in the tables). Column 2 and 3 give the number of plantations and factories, respectively, whereas in Columns 4 and 5 the total weight of the items (plantation production) and the total sum of the capacities of the factories are shown.

In contrast to the other two applications, different knapsacks have different capacities. Moreover, the weights of the items (the production of the plantations) cover a wide range of numbers. These instances appear to be more difficult for our

Problem	$\|N\|$	$\|M\|$	$\sum_{i \in N} f_i$	$\sum_{k \in M} F_k$
sa1	450	4	97714	106346
sa2	370	4	82914	97224

Table 2.8: Sugar cane alcohol production: Problem data

code than the previous ones and we cannot solve these problems without branching. Our results are summarized in Table 2.9. About 20% of the CPU time was spent in the separation routines.

Problem	Opt. Sol.	Ind. Ineq.	Joint Ineq.	B&B	CPU Time
sa1	14592921	110	14	121	57:28.68
sa2	11733362	15	9	48	19:33.73

Table 2.9: Results for problems from sugar cane alcohol production

The number of cutting planes compared to the number of branch and cut nodes is quite small. So, the question may arise, why not just branch and bound? However, by adding no cutting planes we were not able to prove optimality within several hundreds of branch and bound nodes. Thus, the individual and joint inequalities (though quite few) are important to solve these instances in reasonable time. Certainly, the numbers also indicate that there is still a lack of good inequalities for these instances.

An interesting peculiarity of most practical problem instances is that the first lower bound is quite close to the value of the optimal solution. The reason for this is that, for many items $i \in N$, the objective function coefficients c_{ik}, $k \in M$, are similar. However, the gap between the lower bound and the upper bound after the first iteration is larger by far. Only when individual and especially joint inequalities are added, the linear programming solution provides structural information such that the primal heuristics find good upper bounds or even an optimal solution. This fact can also be observed by running random examples.

$\|N\|$	$\|M\|$	Average Gap after first LP	Average Gap after Indiv. Ineq.		Average Gap after Joint Ineq.	
50	4	561.0	231.4	(41.2%)	36.6	(6.5%)
100	4	265.4	164.0	(61.8%)	78.6	(29.6%)
150	4	355.2	149.2	(42.0%)	114.2	(32.3%)
200	4	335.4	117.0	(34.9%)	75.2	(22.4%)
300	4	332.0	112.4	(33.9%)	100.6	(30.3%)
400	4	210.5	154.8	(73.5%)	90.0	(42.8%)
500	4	171.0	41.5	(24.3%)	38.2	(22.4%)

Table 2.10: Impact of individual and joint inequalities on random examples

Table 2.10 provides typical results for random problems. The first two columns give the number of items and the number of knapsacks. The weights of the items are randomly chosen from the interval $[5, 300]$, the objective function coefficients are random numbers in the interval $[1, 1000]$, and the knapsack capacities are randomly computed such that $\sum_{k \in M} F_k \leq \alpha \sum_{i \in N} f_i$, where α is a random number chosen from $[1.05, 1.3]$. We have created four different problems with the same number of knapsacks and items. The numbers in Column 3 to 5 show the average over the absolute values of the gaps between the upper bounds and the lower bounds after the first iteration, after no further violated individual inequalities were found, and after no further violated joint inequalities were found. The number in brackets give the improvement of the gap in percentage. Although we cannot solve most of these random examples to optimality without branching, the results confirm that the gap between the lower and an upper bound is substantially decreased by using individual and joint inequalities.

Chapter 3

The Steiner Tree Packing Problem

3.1 Introduction

The type of problems we consider in this chapter lead to integer programs with block structure, where the blocks consist of certain set covering problems and where the linking constraints are of set packing type as in Chapter 2. As we will see, this constellation gives rise to new problems and questions. For example, it is no longer easy to determine a feasible solution. This fact has a strong impact on the study of the associated polyhedron and its facet-defining inequalities. We start by defining the problem we are going to consider.

Given a graph $G = (V, E)$ and a node set $T \subseteq V$, we call an edge set $S \subseteq E$ a *Steiner tree for* T if, for each pair of nodes $s, t \in T$, the graph $(V(S), S)$ contains a path from s to t. In this chapter we investigate the following problem that we call the *(weighted) Steiner tree packing problem.*

Problem 3.1.1 (The weighted Steiner tree packing problem)

Instance:

A graph $G = (V, E)$ *with positive, integer capacities* $c_e \in \mathbb{N}$ *and non-negative weights* $w_e \in \mathbb{R}_+$, $e \in E$.
A list of node sets $\mathcal{N} = \{T_1, \ldots, T_N\}$, $N \geq 1$, *with* $T_k \subseteq V$ *for all* $k = 1, \ldots, N$.

Problem:

Find edge sets $S_1, \ldots, S_N \subseteq E$ *such that*

(i) S_k *is a Steiner tree in* G *for* T_k *for all* $k = 1, \ldots, N$,

(ii) $\displaystyle\sum_{k=1}^{N} |S_k \cap \{e\}| \leq c_e$ *for all* $e \in E$,

(iii) $\displaystyle\sum_{k=1}^{N} \sum_{e \in S_k} w_e$ *is minimal.*

If requirement (iii) in Problem 3.1.1 is omitted we call the corresponding problem the *Steiner tree packing problem* without the prefix "weighted". A feasible solution of Problem 3.1.1 is a collection of Steiner trees $S_1, \ldots, S_N$ that satisfy (i) and (ii). It is convenient to order the sets S_k and call an N-tuple $(S_1, \ldots, S_N)$ of edge sets a *Steiner tree packing* or *packing of Steiner trees* if the sets $S_1, \ldots, S_N$ form a feasible solution of the Steiner tree packing problem.

We denote by $\mathrm{STP}_w(G, \mathcal{N}, c, w)$ an instance of the (weighted) Steiner tree packing problem, i.e., a graph $G = (V, E)$, a net list $\mathcal{N} = \{T_1, \dots, T_N\}$, $N \in \mathbb{N}$, edge capacities $c_e, e \in E$, and edge weights $w_e, e \in E$. If we neglect the edge weights and consider the Steiner tree packing problem as a feasibility problem, we denote an instance by $\mathrm{STP}(G, \mathcal{N}, c)$. In the application we have in mind it is usual to call the list of node sets $\mathcal{N}$ a *net list*. We follow this custom. The number N denotes the cardinality of the net list. Any element $T_k \in \mathcal{N}$ is called a *set of terminals* or a *net* and the nodes $t \in T_k$ are called *terminals*. Instead of net T_k we will often simply say *net* k. To avoid the discussion of (trivial) special cases we assume throughout this chapter that every terminal set of a net list $\mathcal{N}$ has at least cardinality two and that $N \geq 1$.

Our definition of a Steiner tree slightly differs from the terminology most frequently used in the literature. A Steiner tree is usually supposed to be a tree. Our definition, however, simplifies notation and is more convenient for the polyhedral investigations in the following. A Steiner tree that is a tree and whose leaves are terminals is called *edge-minimal*. Accordingly, a Steiner tree packing $(S_1, \dots, S_N)$ is called *edge-minimal*, if each S_k is edge-minimal.

It is not surprising that the Steiner tree packing problem and its weighted form are $\mathcal{NP}$-complete or $\mathcal{NP}$-hard, respectively, even in special cases. For example, the following variants are hard.

If we restrict the weighted Steiner tree packing problem to $N = 1$ and $c_e = 1$, for all $e \in E$, we obtain the problem of finding a minimal Steiner tree in G. This problem is $\mathcal{NP}$-hard even if G is restricted to be planar or a grid graph (Karp [1972], Garey and Johnson [1977]). Futhermore, it is $\mathcal{NP}$-complete to decide whether there exists a feasible solution for the Steiner tree packing problem. Results here are due to Kramer and van Leeuwen [1984], who proved that the problem of finding N edge-disjoint paths is $\mathcal{NP}$-complete. Similarly, it was shown in Korte, Prömel, and Steger [1990] that it is $\mathcal{NP}$-complete to decide whether a packing of two Steiner trees exists.

Let us now present an integer programming formulation for the weighted Steiner tree packing problem $\mathrm{STP}_w(G, \mathcal{N}, c, w)$, given by a graph $G = (V, E)$ with edge capacities $c_e \in \mathbb{N}, e \in E$, and weights $w_e, e \in E$, and a net list $\mathcal{N} = \{T_1, \dots, T_N\}$. Besides the notation in Appendix A we use the following. We consider the $N \cdot |E|$ – dimensional vector space $\mathbb{R}^E \times \dots \times \mathbb{R}^E$ which we abbreviate by $\mathbb{R}^{\mathcal{N} \times E}$. The components of a vector $x \in \mathbb{R}^{\mathcal{N} \times E}$ are indexed by x_e^k for $k \in \{1, \dots, N\}, e \in E$. For a vector $x \in \mathbb{R}^{\mathcal{N} \times E}$ and $k \in \{1, \dots, N\}$ we denote by $x^k \in \mathbb{R}^E$ the vector $(x_e^k)_{e \in E}$. Instead of $x = ((x^1)^T, \dots, (x^N)^T)^T \in \mathbb{R}^{\mathcal{N} \times E}$ we often write $x = (x^1, \dots, x^N)$ if the meaning of the symbols is clear from the context. We call the vector $(\chi^{S_1}, \dots, \chi^{S_N}) \in \mathbb{R}^{\mathcal{N} \times E}$ the *incidence vector of a Steiner tree packing* $P = (S_1, \dots, S_N)$. We will often abreviate the incidence vector of a Steiner tree packing P by χ^P. Consider the following integer program.

$$
(3.1) \qquad
\begin{array}{lll}
\min & \displaystyle\sum_{k=1}^{N} \sum_{e \in E} w_e x_e^k & \\
(i) & x^k(\delta(W)) \geq 1, & \text{for all } W \subset V,\ W \cap T_k \neq \emptyset,\ (V \setminus W) \cap T_k \neq \emptyset, \\
& & k = 1, \dots, N. \\
(ii) & \displaystyle\sum_{k=1}^{N} x_e^k \leq c_e, & \text{for all } e \in E. \\
(iii) & 0 \leq x_e^k \leq 1, & \text{for all } e \in E,\ k = 1, \dots, N. \\
(iv) & x_e^k \in \{0, 1\}, & \text{for all } e \in E,\ k = 1, \dots, N.
\end{array}
$$

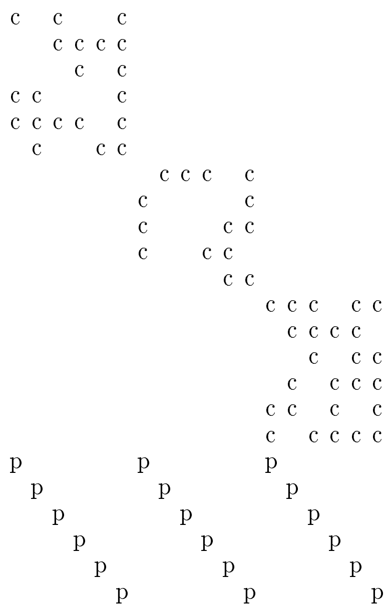


Figure 3.1: Structure of a Steiner tree packing integer program with $|E| = 6$ and $N = 3$: The letter 'c' indicates a non-zero entry in a Steiner cut inequality, which is a special set covering constraint. The letter 'p' shows a non-zero entry in the capacity constraints, which are special set packing inequalities.

The inequalities (3.1) (i) are the so-called *Steiner cut inequalities*. The inequalities (3.1) (ii) are called *capacity inequalities* and the ones in (3.1) (iii) *trivial inequalities*.

Obviously, each incidence vector of a Steiner tree packing satisfies (3.1) (i) – (iv) and vice versa, each vector $x \in \mathbb{R}^{\mathcal{N} \times E}$ satisfying (3.1) (i) – (iv) is the incidence vector of a Steiner tree packing. Thus, the weighted Steiner tree packing problem can be solved via (3.1). Figure 3.1 shows the structure of such an integer program.

Let us define the *Steiner tree packing polyhedron*, denoted by $P_{\mathrm{STP}}(G, \mathcal{N}, c)$, to be the convex hull of all incidence vectors of Steiner tree packings. We have

$$(3.2) \qquad P_{\mathrm{STP}}(G, \mathcal{N}, c) = \mathrm{conv}\{x \in \mathbb{R}^{\mathcal{N} \times E} : x \text{ satisfies (3.1) (i),\ldots,(iv)}\}.$$

If $N = 1$ and $c = \mathbb{1}$, i.e., $c_e = 1$ for all $e \in E$, we also refer to $P_{\mathrm{STP}}(G, \{T_1\}, \mathbb{1})$ as the *Steiner tree polyhedron*. The aim of the following two sections will be to study $P_{\mathrm{STP}}(G, \mathcal{N}, c)$ and to describe this polyhedron partially by valid and facet-defining inequalities.

The motivation for studying the Steiner tree packing problem and the polytope $P_{\mathrm{STP}}(G, \mathcal{N}, c)$ arises from the design of electronic circuits. The design of electronic circuits is a hierarchical process consisting of several phases. The beginning is a description of the task the circuit to be designed must perform. Such a task can be viewed as a complex logical function that consists of many elementary logic operations. Usually several of these elementary logic operations are combined into a logical unit (for example an adder). In the *logical design phase* chip designers specify which of these predefined logical units are to be used, and determine which of the chosen logical units must be connected by wires so that the chip performs in the way it is supposed to.

The logical units are also called *cells*. Each cell is characterized by its width, its height, its contact points (so-called *terminals*) and its electric properties. A *net* is a set of terminals that must be connected by a wire (as specified in the logical design phase). The list of cells and the list of nets are the input of the second phase, the *physical design*. Here, the task is to assign the cells to a certain rectangular area and connect (route) the nets by wires. The physical design problem is, of course, more complicated than the sketch above suggests, since certain design rules have to be taken into account, an objective function is to be minimized, etc. The design rules strongly depend on the given layout style and specify, for instance, the distance two nets must stay apart, whether certain cells are preassigned to certain locations and so on. This applies especially to the objective function. Usually, the primary goal is to minimize the whole area of the chip or, if the chip area is fixed in advance, to guarantee routability, i. e., to solve the problem of placing the cells on the chip such that there exists a feasible solution to the routing problem.

However, routability can hardly be measured and expressed in terms of an objective function. Thus, minimizing the total length of all routes is very often used instead. Another reason for minimizing the routing length is that an electronic circuit with small routing length usually needs little area on the whole. Thus, minimizing the overall area is (somehow) implicitly taken into account by minimizing the routing length.

Any reasonably precise version of the physical design problem is $\mathcal{NP}$-hard, even very simple models are. Moreover, most real-world problem instances involve several thousands of cells and nets, so that today's algorithmic knowledge makes it very improbable that they can be solved to optimality. Therefore, the physical design problem is (heuristically) decomposed into subproblems. The first subproblem typically consists of finding appropriate locations for the cells (*placement problem*). Subsequently, the nets must be realized by wiring the appropriate terminals (*routing problem*) and finally, a compaction step is performed if required. This process is iterated with different parameters if the final result is not satisfactory.

We are interested in the routing problem. That is, we are given a list of nets. Each net consists of a set of terminals. The terminals specify the points at which wires have to contact the cells. The routing problem is to connect the nets by wires on the routing area subject to certain technical side constraints. As mentioned above, the objective usually is to minimize the overall wiring length.

The routing itself takes place on so-called *layers*. Each layer is divided into *tracks* on which the wires run. The tracks and the *vias*, the points where wires change the layers, must meet certain distance requirements. In practice, the routing problem itself is again decomposed because of its inherent complexity and large scale. In the *global routing phase* the homotopy of the nets is determined, i. e., it is determined how the wires "maneuver around the cells". Thereafter, in the *detailed routing phase* the wires are assigned to the layers and tracks according to the homotopy specified in the global routing step. This decomposition scheme gives rise to many variants of the routing problem.

A number of the routing problems resulting from this approach can be modeled as a (weighted) Steiner tree packing problem. We will illustrate two examples in the following.

For modeling the global routing problem, the routing area is subdivided into subareas. This is done in a way such that the resulting subareas have certain special properties, for instance, they contain no holes (i. e., there are no cells located within the areas) or they have simple shapes (for example, rectangles). These subareas are represented by the nodes or the edges of some graph. We describe the node representation. Here, two nodes are connected by an edge, if the corresponding subareas are adjacent. Let $G = (V, E)$ denote the resulting graph. Additionally, a capacity $c_{uv} \in \mathbb{N}$ is assigned to an edge $uv \in E$ limiting the number of nets that may run between the subareas associated with the two endnodes of this edge. The weight of an edge w_{uv} corresponds to the distance between the two midpoints of the according subareas. Every terminal of a net is assigned to that node, whose corresponding subarea contains the terminal or is closest to the position of the terminal. The global routing problem consists in routing all nets in G such that the capacity constraints are satisfied and the total wiring length is as small as possible. Obviously, this task defines an instance of the weighted Steiner tree packing problem.

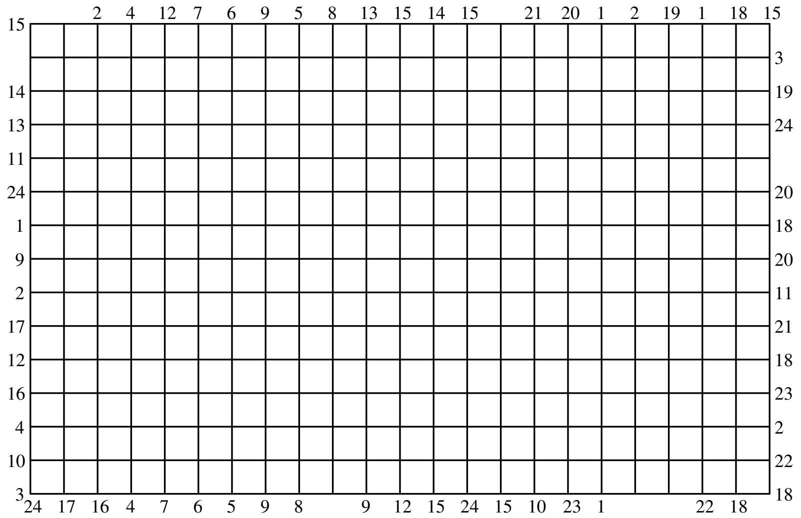


Figure 3.2: A switchbox routing problem

After having solved the global routing problem every subarea that corresponds to a node in the global routing graph must be routed in detail. The number of different detailed routing models which are studied in the literature or which are used in practice is tremendous. We want to focus on one variant of the detailed routing problem, called *switchbox routing problem* (see Figure 3.2). Here, the underlying graph is a complete rectangular grid graph and the terminal sets are located on the four sides of the grid. Remember that the task of detailed routing is to assign the wires to layers and tracks. Detailled routing problems, and thus also switchbox routing problems, are classified by distinguishing whether or to which extent the layers are taken into account while the nets are assigned to tracks. Here, the following models are of special interest.

Multiple layer model Given a k layered grid graph (that is a graph obtained by stacking k copies of a grid graph on top of each other and connecting related nodes by perpendicular lines), where k denotes the number of layers. The nets have to be routed in a node disjoint fashion. The multiple layer model is well suited to reflect reality. The disadvantage is that in general the resulting graphs are very large.

Manhattan model Given a (subgraph of a) complete rectangular grid graph. The nets must be routed in an edge disjoint fashion with the additional restriction that nets that meet at some node are not allowed to bend at this node, i. e., so-called *knock-knees* (cf. Figure 3.3) are not allowed. This restriction guarantees that the resulting routing can be realized on two layers at the possible expense of causing long detours.

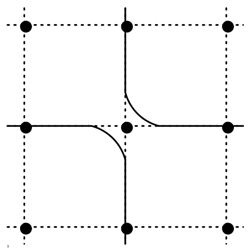


Figure 3.3: A knock-knee

Knock-knee model Again, a (subgraph of a) complete rectangular grid graph is given and the task is to find an edge disjoint routing of the nets. In this model knock-knees are possible. As we will see in Section 3.5, the wiring length of a solution in this case is sometimes smaller than in the Manhattan model. The main drawback is that the assignment to layers is neglected. Brady and Brown [1984] have designed an algorithm that guarantees that any solution in this model can be routed on four layers. It was shown in Lipski [1984] that it is $\mathcal{NP}$-complete to decide whether a realization on three layers is possible.

As in the case of the global routing problem the weighted Steiner tree packing problem is a natural mathematical model of the switchbox routing problem in the knock-knee mode. As we will see later in this chapter, also the switchbox routing problem in the Manhattan model may be modeled as a weighted Steiner tree packing problem by adding additional linear constraints. All examples that this computational study reports on are instances of these two types of switchbox routing problems. Both problems are $\mathcal{NP}$-complete (Sarrafzadeh [1987], Szymanski [1985]). Of course, we only sketched some of the routing problems arising in the design of electronic circuits and that can be modeled as Steiner tree packing problems. For more details on this subject we refer to the book of Lengauer [1990].

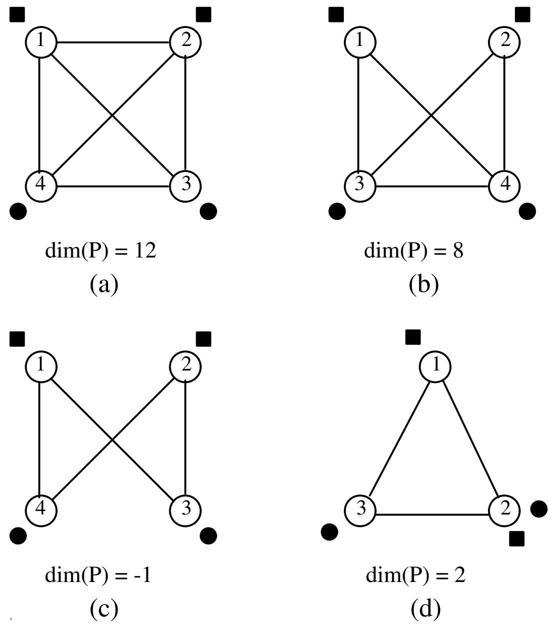
3.2 The Steiner Tree Packing Polyhedron: Basic Results

In this section we discuss some basic properties of the Steiner tree packing polyhedron, including a discussion of its dimension, the manipulation of valid inequalities, the lifting of single Steiner tree inequalities and a discussion of the trivial and capacity inequalities.

Let us first consider the dimension. Since it is already $\mathcal{NP}$-complete to decide whether the Steiner tree packing problem has a feasible solution (Kramer and van Leeuwen [1984], Korte, Prömel, and Steger [1990]), it is obviously also $\mathcal{NP}$-complete to decide the dimension of Steiner tree packing polyhedron.

This result does not give much hope for a successful study of Steiner tree packing polyhedra of general instances STP $(G, \mathcal{N}, c)$. Figure 3.4 shows some examples and the corresponding dimensions. The affine hull of the polytope of Figure 3.4 (b) is given by $x^1_{34} = 0$, $x^2_{34} = 1$; that of the polytope of Figure 3.4 (d) by $x^1_{12} = 1, x^2_{12} = 0, x^1_{23} = 0, x^2_{23} = 1$, for instance. The dimension jumps appear to be rather erratic.

We have decided to study the Steiner tree packing polyhedron for special problem instances for which the dimension can be determined easily and to look for facet-defining inequalities for these special instances. Clearly, such an approach is only sensible if the results can be carried over (at least partially) to practically interesting instances as they occur, for example, in the design of electronic circuits.



Figures (a) to (d) show some examples and the dimension of the corresponding polyhedron. The two terminal sets are drawn as rectangles or cycles respectively ($T_1=\{1,2\}$, $T_2=\{3,4\}$ or $T_2=\{2,3\}$ resp.) and P abbreviates $P_{\mathrm{STP}}(G,\mathcal{N},\mathbb{1})$. The polyhedron in (a) is full dimensional. Deleting the edge with endnodes 1 and 2 (Figure (b)) decreases the dimension by 4. If additionally the edge connecting nodes 3 and 4 (Figure (c)) is deleted, there even does not exist any feasible solution. Figure (d) shows an example in which the underlying graph is complete but the corresponding polyhedron is not full dimensional.

Figure 3.4: Dimensions of Steiner tree packing polyhedra

It has turned out that an instance STP $(G,\mathcal{N},c)$, where the graph G is complete, the net list $\mathcal{N} = \{T_1,\dots,T_N\}$ is *disjoint*, i.e., $T_i \cap T_j = \emptyset$ for all $i,j \in \{1,\dots,N\}$, $i \neq j$, and the capacities are equal to one ($c = \mathbb{1}$), is a suitable case. The following lemma shows that the Steiner tree packing polyhedron is full dimensional in this case.

Lemma 3.2.1 *Let $G = (V,E)$ be the complete graph with node set $V, |V| \geq 3$, and edge capacities $c_e = 1$, $e \in E$. Furthermore let $\mathcal{N} = \{T_1,\dots,T_N\}$ be a disjoint net list with $T_1,\dots,T_N \subseteq V$. Then,*

$$dim\,(P_{STP}\,(G,\mathcal{N},c)) = N \cdot |E|.$$

Proof. Let λ be a vector with $\lambda^T x = 0$ for all $x \in \operatorname{diff}(P_{\mathrm{STP}}\,(G,\mathcal{N},c))$. We have to show that $\lambda_e^k = 0$ for all $e \in E$ and $k \in \{1,\dots,N\}$. Let $e \in E$ be an arbitrary edge with endnodes u and v. We choose Steiner trees S_k, $k \in \{1,\dots,N\}$, as follows. If $e \in E(T_k)$, set $S_k = [t : T_k]$ for some $t \in V \setminus \{u,v\}$. Such a node t exists since $|V| \geq 3$. Otherwise, set $S_k = E(T_k)$. Since $\mathcal{N}$ is a disjoint net list, $P = (S_1,\dots,S_N)$ defines a packing of Steiner trees with $e \notin P$. Thus, $P' = (S_1,\dots,S_k \cup \{e\},\dots,S_N)$ is a Steiner tree packing as well and we get $\lambda_e^k = \lambda^T(\chi^{P'} - \chi^P) = 0$. □

In order to, at least partially, carry over results for the special instance described in Lemma 3.2.1 to any problem instance, we have to discuss how inequalities must be modified if the underlying graph is manipulated using operations such as node splitting or addition, deletion or contraction of an edge. The following lemma summarizes the results of Grötschel, Martin, and Weismantel [1996a].

Lemma 3.2.2 *Consider an instance $STP(G, \mathcal{N}, c)$ of the Steiner tree packing problem and let $a^T x \geq \alpha$ be a valid or facet-defining inequality for $P_{STP}(G, \mathcal{N}, c)$.*

Deleting an edge. *Deletion of an edge preserves validity. However, if $a^T x \geq \alpha$ is facet-defining the new inequality after deletion need not be facet-defining as well.*

Adding an edge. *Adding an edge preserves validity and the property of being facet-defining if the variables that correspond to the new edge are exactly lifted (for a discussion of lifting, see Appendix C).*

Splitting a node. *If we split a node and assign coefficients of zero to the new edge for all nets, the inequality stays valid. However, as in the case of deletion, it need not be facet-defining (even if we determine the coefficients by exact lifting).*

Contracting an edge. *In this case, the new inequality (that is obtained after deleting all coefficients corresponding to the contracted edge) need not be valid any more.*

We have seen in Chapter 2 that single knapsack inequalities also define facets for the multiple knapsack polyhedron. The proof of this result is rather straightforward. Under some mild assumption the same can be shown for the Steiner tree inequalities.

Theorem 3.2.3 *Let $G = (V, E)$ be the complete graph with node set V and $\mathcal{N} = \{T_1, \dots, T_N\}$, $N \geq 2$, a disjoint net list. Let $\bar{a}^T x \geq \alpha$, $\bar{a} \in \mathbb{R}^E$, be a non-trivial facet-defining inequality for $P_{STP}(G, \{T_1\}, \mathbb{1})$. Then, $a^T x \geq \alpha$ defines a facet of $P_{STP}(G, \mathcal{N}, \mathbb{1})$, where $a \in \mathbb{R}^{\mathcal{N} \times E}$ denotes the vector with $a_e^1 = \bar{a}_e^1$, $a_e^k = 0$ for all $k = 2, \dots, N$, $e \in E$.*

The proof of Theorem 3.2.3 is very technical and by far not obvious and we refer the interested reader to Grötschel, Martin, and Weismantel [1996a]. As in the case of the multiple knapsack problem this result implies that, in order to obtain a complete characterization of some Steiner tree packing polyhedron $P_{\text{STP}}(G, \mathcal{N}, c)$, for all nets of the net list, all individual Steiner tree polyhedra $P_{\text{STP}}(G, \{T\}, c), T \in \mathcal{N}$, must be known completely. Note that there are examples where the trivial inequalities define facets for $P_{\text{STP}}(G, \{T\}, c)$, but not for $P_{\text{STP}}(G, \mathcal{N}, c)$. The following theorem gives necessary and sufficient conditions for the trivial and capacity inequalities to be facet-defining in case $N \geq 2$ (the case $N = 1$ was solved in Grötschel and Monma [1990]).

Theorem 3.2.4 *Let $STP(G, \mathcal{N}, c)$ be a Steiner tree packing instance where $\mathcal{N}$ is disjoint and G is complete. Let $e \in E$ and $k \in \{1, \dots, N\}$. Then,*

(i) the inequality $x_e^k \geq 0$ defines a facet of $P_{STP}(G, \mathcal{N}, c)$ if and only if $|V| \geq 5$ or $e \notin E(T_k)$;

(ii) the inequality $x_e^k \leq 1$ defines a facet of $P_{STP}(G, \mathcal{N}, c)$ if and only if $c_e \geq 2$;

(iii) the inequality $\sum_{k=1}^{N} x_e^k \leq c_e$ defines a facet of $P_{STP}(G, \mathcal{N}, c)$ if and only if $c_e \leq N - 1$.

3.3 Joint Inequalities

In this section we consider inequalities that combine two or more nets. We will proceed in the following way. First, we describe each inequality. All inequalities we are going to consider are of the form $a^T x \geq \alpha$, $a \geq 0$. The coefficients of some of the edges will turn out to be zero for all nets. We call these edges *zero edges* and the graph induced by the zero edges the *zero graph*. We will use the structure of the zero graph to name the inequalities. This has the following reasons. The zero graph is structured in such a way that there exists no Steiner tree packing for the nets involved in this graph. Therefore, each feasible solution must use edges whose coefficients are different from zero. This means that each inequality is in some sense (but not necessarily uniquely) determined by the zero graph.

We will always define the inequalities for an arbitrary instance without guaranteeing that the inequality is also valid for the corresponding polyhedron. In the subsequent theorem we characterize the instances for which the inequality is valid. Here, additional edges get value zero for some single nets (we typically denote these sets by $F_1, \ldots, F_N$). In order to show that the inequalities are also facet-defining these edge sets $F_1, \ldots, F_N$ must usually satisfy very technical restrictions. The results can often be generalized, for example, by modifying the net list or by adding a node. In this section we concentrate on the validity of the corresponding inequalities at the expense of giving proofs that the inequalities are facet-defining. In particular, the proof that the corresponding inequalities are facet-defining requires essentially the same scheme. We illustrate this scheme on one sample. For specific proofs of the remaining statements we refer the interested reader to Martin [1992].

3.3.1 Alternating Cycle Inequalities

Definition 3.3.1 *Let $G = (V, E)$ be a graph and $\mathcal{N} = \{T_1, T_2\}$ a net list. We call a cycle F an* alternating cycle *with respect to T_1, T_2, if $F \subseteq [T_1 : T_2]$ and $V(F) \cap T_1 \cap T_2 = \emptyset$ (see Figure 3.5). Moreover, let $F_1 \subseteq E(T_2)$ and $F_2 \subseteq E(T_1)$ be two sets of diagonals of the alternating cycle F with respect to T_1, T_2. The inequality*

$$(\chi^{E \setminus (F \cup F_1)}, \chi^{E \setminus (F \cup F_2)})^T x \geq \tfrac{1}{2}|F| - 1$$

is called an alternating cycle inequality.

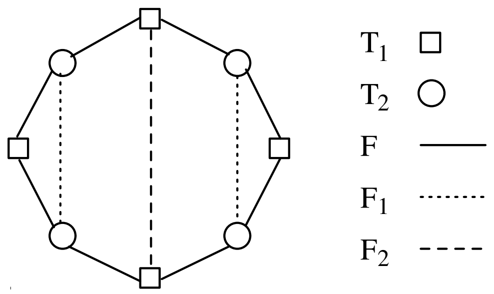


Figure 3.5: Illustration of an alternating cycle inequality

It is not difficult to see that the basic form of an alternating cycle inequality, i.e., $F_1 = F_2 = \emptyset$, is valid for $P_{\mathrm{STP}}(G, \mathcal{N}, \mathbb{1})$, but in general, it is not facet-defining. The sets F_1 and F_2 are used to strengthen the basic form; in fact, choosing them appropriately we can obtain facet-defining inequalities.

The sets of diagonals $F_1 \subseteq E(T_2)$ and $F_2 \subseteq E(T_1)$ are called *maximal cross free with respect to F*, if F_1 and F_2 are cross free (for a definition of cross free, see Appendix A), and each diagonal $e_1 \in E(T_1) \setminus F_2$ crosses F_1 and each diagonal $e_2 \in E(T_2) \setminus F_1$ crosses F_2 (see Figure 3.5 for an example). Then, the following theorem holds.

Theorem 3.3.2 *Let $G = (V, E)$ be the complete graph with node set V and let $\mathcal{N} = \{T_1, T_2\}$ be a disjoint net list with $T_1 \cup T_2 = V$ and $|T_1| = |T_2| = l$, $l \geq 2$. Furthermore, let F be an alternating cycle with respect to T_1, T_2 with $V(F) = V$ and $F_1 \subseteq E(T_2)$, $F_2 \subseteq E(T_1)$. Then, the alternating cycle inequality*

$$(\chi^{E\setminus(F\cup F_1)}, \chi^{E\setminus(F\cup F_2)})^T x \geq l - 1$$

is valid for $P_{STP}(G, \mathcal{N}, \mathbb{1})$ if and only if F_1 and F_2 are cross free. It is facet-defining if and only if F_1 and F_2 are maximal cross free.

Proof. Set $E_k = E \setminus (F \cup F_k)$, $k = 1, 2$ and $a = (\chi^{E_1}, \chi^{E_2})$. First, we prove that $a^T x \geq l - 1$ is valid if F_1 and F_2 are cross free. It suffices to show that for every packing of Steiner trees (S_1, S_2), $|(S_1 \cap E_1) \cup (S_2 \cap E_2)| \geq l - 1$ holds (note that $c = \mathbb{1}$).

Let (S_1, S_2) be any Steiner tree packing. W.l.o.g. S_1 and S_2 are edge-minimal. Set $T_1' = \{t \in T_1 : \delta(t) \cap F \subseteq S_2\}$ and $T_2' = \{t \in T_2 : \delta(t) \cap F \subseteq S_1\}$. Since S_1 and S_2 are edge-minimal and $|F| = 2l$, we have that $|T_1'| + |T_2'| \leq l - 1$. This implies that $T_1 \setminus T_1'$ and $T_2 \setminus T_2'$ are non-empty. Therefore, at least $|T_1'| + |T_2'|$ edges $e \in S_1 \cap E_1 \cup S_2 \cap E_2$ are necessary to connect T_1' with $T_1 \setminus T_1'$ and T_2' with $T_2 \setminus T_2'$. Consider the remaining terminals $T_1 \setminus T_1'$ and $T_2 \setminus T_2'$. Set $k_i = \kappa\big((V(S_i), S_i \setminus F_i)\big)$ for $i = 1, 2$, where $\kappa(\hat{G})$ denotes the number of components of graph $\hat{G}$. Since F_1 and F_2 are cross free, we obtain $k_1 + k_2 \leq l + 1$. Thus,

$$\begin{aligned} a^T(\chi^{S_1}, \chi^{S_2}) \quad &\geq (|T_1'| + |T_2'|) + (|T_1 \setminus T_1'| + |T_2 \setminus T_2'| - (k_1 + k_2)) \\ &\geq |T_1| + |T_2| - (k_1 + k_2) \geq l - 1. \end{aligned}$$

Let us now outline the proof that $a^T x \geq l - 1$ defines a facet of $P_{\text{STP}}(G, \mathcal{N}, \mathbb{1})$.

Suppose $b^T x \geq \beta$ is a facet-defining inequality of $P_{\text{STP}}(G, \mathcal{N}, \mathbb{1})$ such that $F_a = \{x \in P_{\text{STP}}(G, \mathcal{N}, \mathbb{1}) : a^T x = l - 1\} \subseteq F_b = \{x \in P_{\text{STP}}(G, \mathcal{N}, \mathbb{1}) : b^T x = \beta\}$. In the following we show that b is a multiple of a.

In the first two steps we show that for any coefficient $a_e^k = 0$, $k \in \{1, 2\}$ there exists a Steiner tree packing P with $a^T \chi^P = l - 1$ and $e \notin P$. This implies $b_e^k = 0$.

(1) $b_e^k = 0$ for $e \in F$, $k = 1, 2$.

Choose $S_1 = F \setminus \{e\}$ and $S_2 = [t : T_2]$, $t \in T_2$. Furthermore set $S_1' = S_1 \cup \{e\}$. Then $P = (S_1, S_2)$ and $P' = (S_1', S_2)$ are Steiner tree packings with $\chi^P, \chi^{P'} \in F_a$ and $0 = b^T(\chi^{S_1'}, \chi^{S_2}) - b^T(\chi^{S_1}, \chi^{S_2}) = b_e^1$. Analogously we obtain $b_e^2 = 0$.

(2) $b_e^k = 0$ for $e \in F_k$, $k = 1, 2$.

Choose $S_1 = F$ and $S_2 = [t : T_2]$, $t \in T_2$. Furthermore set $S_1' = S_1 \cup \{e\}$. Then $P = (S_1, S_2)$ and $P' = (S_1', S_2)$ are Steiner tree packings with $\chi^P, \chi^{P'} \in F_a$ and $0 = b^T(\chi^{S_1'}, \chi^{S_2}) - b^T(\chi^{S_1}, \chi^{S_2}) = b_e^1$. Analogously we obtain $b_e^2 = 0$.

Next, we prove that the coefficients of edges that connect terminals of the same net are equal. This is done by constructing two Steiner trees inside the subgraph induced by the corresponding terminal set that differ only in two edges.

(3) $b_e^k = b_{e'}^k$ for $e, e' \in E(T_k)$, $k = 1, 2$.

Let $e = uv$ with $u, v \in T_1$. Set $S_2 = F$ and $S_1 = [v : T_1]$. Let $e' \in [u : T_1] \setminus \{e\}$ and $S_1' = S_1 \setminus \{e\} \cup \{e'\}$. Then $P = (S_1, S_2)$ and $P' = (S_1', S_2)$ are Steiner tree packings with $\chi^P, \chi^{P'} \in F_a$ and $0 = b^T(\chi^{S_1'}, \chi^{S_2}) - b^T(\chi^{S_1}, \chi^{S_2}) = b^1_{e'} - b^1_e$ for all $e, e' \in \delta(u)$, $u \in T_1$. Analogously we obtain $b^2_e = b^2_{e'}$.

In the remainder of the proof set $\bar{k} = 1$, if $k = 2$, and $\bar{k} = 2$, if $k = 1$.

In steps (4) and (5) we fix the remaining coefficients of one net. To this end we use the structure of the zero graph, the properties fullfilled by F_1 and F_2 and the fact proved in (3).

(4) $b^k_e = b^k_{e'}$ for $e' \in E(T_k)$, $e \in [T_k : T_{\bar{k}}]$, $k = 1, 2$.

Let $e = uw$ with $u \in T_1$, $w \in T_2$ and $v \in T_1$ such that $vw \in F$. Choose $S_2 = F \setminus \delta(v)$, $S_1 = [u : T_1]$ and $S_1' = S_1 \setminus \{uv\} \cup \{uw\} \cup \{vw\}$. Then $P = (S_1, S_2)$ and $P' = (S_1', S_2)$ are Steiner tree packings with $\chi^P, \chi^{P'} \in F_a$ and $0 = b^T(\chi^{S_1'}, \chi^{S_2}) - b^T(\chi^{S_1}, \chi^{S_2}) = b^1_{uw} + b^1_{vw} - b^1_{uv} = b^1_{uw} - b^1_{uv}$, because $b^1_{vw} = 0$ (see (1)). This together with (3) proves the statement. Analogously we obtain $b^2_e = b^2_{e'}$.

(5) $b^k_e = b^k_{e'}$ for $e \in E(T_{\bar{k}}) \setminus F_k$, $e' \in E(T_k)$, $k = 1, 2$.

Let $e = uv \in E(T_2) \setminus F_1$. Since F_1 and F_2 are maximal cross free, there exists an edge $u_2v_2 \in F_2$ which crosses e. Let u^-, $v^+ \in T_1$ such that u^-u, $vv^+ \in F$ and uv crosses u^-v^+. Choose $S_1 = [u^- : T_1]$ and $S_2 = F$. Furthermore set $S_1' = S_1 \setminus \{u^-v^+\} \cup \{u^-u, uv, vv^+\}$ and $S_2' = S_2 \setminus \{u^-u, vv^+\} \cup \{u_2v_2\}$. Then $P = (S_1, S_2)$ and $P' = (S_1', S_2')$ are Steiner tree packings with $\chi^P, \chi^{P'} \in F_a$ and $0 = b^T(\chi^{S_1}, \chi^{S_2}) - b^T(\chi^{S_1'}, \chi^{S_2'}) = b^1_{u^-v^+} - b^1_{uv}$. This together with (3) proves the statement. Analogously we obtain $b^2_e = b^2_{e'}$.

It remains to be shown that the coefficients of different nets are equal. We prove this by constructing two Steiner tree packings; in the first solution the Steiner tree for net 1 uses only zero edges, whereas in the second solution zero edges are only used by net 2.

(6) $b^1_e = b^2_{e'}$ for $e \in E(T_1)$, $e' \in E(T_2)$.

Let $e = uv \in E(T_1)$ and $e' = wx \in E(T_2)$. Choose $S_1 = [u : T_1]$, $S_2 = F$, $S_1' = F$ and $S_2' = [w : E(T_2)]$. Then $P = (S_1, S_2)$ and $P' = (S_1', S_2')$ are Steiner tree packings with $\chi^P, \chi^{P'} \in F_a$ and $0 = b^T(\chi^{S_1'}, \chi^{S_2'}) - b^T(\chi^{S_1}, \chi^{S_2}) = \sum_{i \in T_2 \setminus \{w\}} b^2_{iw} - \sum_{i \in T_1 \setminus \{u\}} b^1_{iu} = (l-1) \cdot b^2_{xw} - (l-1) \cdot b^1_{vu}$ because of (3). So we obtain $b^1_e = b^2_{e'}$.

(1) - (6) imply that b is a multiple of a.

It remains to be shown that F_1 and F_2 are maximal cross free if $a^T x \geq l - 1$ defines a facet of $P_{\text{STP}}(G, \mathcal{N}, \mathbb{1})$.

First, we show that F_1 and F_2 have to be cross free. Suppose, F_1 and F_2 are not cross free. Then, there exist two crossing diagonals $e_1 = u_1v_1 \in F_1$ and $e_2 = u_2v_2 \in F_2$. Let u_1^-, $v_1^+ \in T_1$ such that $u_1^-u_1$, $v_1v_1^+ \in F$ and u_1v_1 crosses $u_1^-v_1^+$. Choose $S_1 = [u_1^- : T_1] \setminus \{u_1^-v_1^+\} \cup \{u_1^-u_1, u_1v_1, v_1v_1^+\}$ and $S_2 = F \setminus \{u_1^-u_1, v_1v_1^+\} \cup \{u_2v_2\}$. Then, (S_1, S_2) is a Steiner tree packing with $a^T(\chi^{S_1}, \chi^{S_2}) = l - 2$, a contradiction.

Finally, we show that F_1 and F_2 are maximal cross free. Suppose, this is not the case. Let $F_1' \subseteq E(T_2)$ and $F_2' \subseteq E(T_1)$ such that $F_1 \cup F_2 \subset F_1' \cup F_2'$ and F_1' and F_2' are maximal cross free. Due to first part of this proof $(\chi^{E \setminus (F \cup F_1')}, \chi^{E \setminus (F \cup F_2')})^T x \geq l - 1$ defines a facet of $P_{\text{STP}}(G, \mathcal{N}, \mathbb{1})$. Summing up this facet-defining inequality together with the valid inequalities $x^1_e \geq 0$ for all $e \in F_1' \setminus F_1$ and $x^2_e \geq 0$ for all $e \in F_2' \setminus F_2$ we obtain $a^T x \geq l - 1$. Thus, $a^T x \geq l - 1$ does not define a facet of $P_{\text{STP}}(G, \mathcal{N}, \mathbb{1})$, a contradiction. □

There are several ways to extend alternating cycle inequalities, for instance by adding parallel edges or adding additional nodes. Depending on whether the new coefficients are lifted sequentially (and in which order) or simultaneously various inequalities come up. We do not describe these possibilities here and refer to Grötschel, Martin, and Weismantel [1996a].

3.3.2 Grid Inequalities

Definition 3.3.3 *Let $G = (V, E)$ be a graph and $\mathcal{N} = \{T_1, T_2\}$ be a net list. Furthermore, let $\hat{G} = (\hat{V}, \hat{E})$ be a subgraph of G such that $\hat{G}$ is a complete rectangular $h \times 2$ grid graph with $h \geq 3$. Assume that the nodes of V are numbered such that $\hat{V} = \{(i, j) : i = 1, \ldots, h, j = 1, 2\}$. Moreover, let $(1,1), (h,2) \in T_1$ and $(1,2), (h,1) \in T_2$. We call the inequality*

$$(\chi^{E \setminus \hat{E}}, \chi^{E \setminus \hat{E}})^T x \geq 1$$

a $h \times 2$ grid inequality.

If we consider in the following a complete rectangular $h \times 2$ grid graph, which is a subgraph of a given graph $G = (V, E)$, we always assume for the ease of notation that the node set V is numbered such that the nodes of the grid graph have a numbering as assumed in Definition 3.3.3.

Theorem 3.3.4 *Let $\hat{G} = (\hat{V}, \hat{E})$ be a complete rectangular $h \times 2$ grid graph with $h \geq 3$. Let J_1 and J_2 be the two columns of $\hat{G}$. Let $\mathcal{N} = \{T_1, T_2\}$ be a net list where $T_1 = \{(1,1), (h,2)\}$ and $T_2 = \{(1,2), (h,1)\}$. Furthermore, let $G = (V, E)$ be a graph with $\hat{V} \subseteq V$, $\hat{E} \subseteq E$ such that $[V(J_1) : V(J_2)]$ is a cut in G. Set $F = \hat{E}$ and let $F_1, F_2 \subset E \setminus F$. Then, the inequality*

$$(\chi^{E \setminus (F \cup F_1)}, \chi^{E \setminus (F \cup F_2)})^T x \geq 1$$

is valid for $P_{STP}(G, \mathcal{N}, \mathbb{1})$ if and only if for all $u, v \in V(F)$, $u \neq v$, there does not exist a path from u to v in (V, F_k) for $k = 1, 2$ (see Figure 3.6).

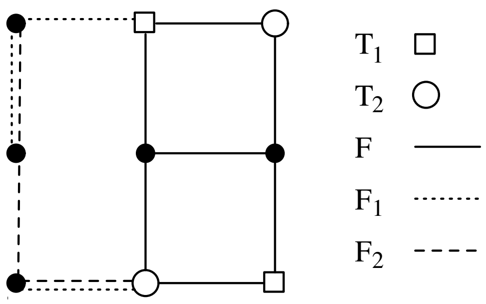


Figure 3.6: Illustration of a 3×2 grid inequality

Proof. The validity of the inequality is easy to see. There obviously does not exist a Steiner tree packing in $(V(F), F)$, since all nodes of $V(F)$ have degree at most three (with respect to F) and the terminal nodes have degree two (with respect to F). Since in addition, for every $u, v \in V(F)$, $u \neq v$ there does not exist a path from u to v in (V, F_k) for $k = 1, 2$, the inequality is valid. On the other hand, if there exist nodes $u, v \in V(F)$, $u \neq v$ and a path from u to v in (V, F_k) for some $k \in \{1, 2\}$, one can easily construct a Steiner tree packing violating the inequality. $\square$

We also worked out necessary and sufficient conditions such that the inequality in Theorem 3.3.4 is facet-defining (see Martin [1992], Grötschel, Martin, and Weismantel [1995]). In Theorem 3.3.4 the underlying graph G does not need to be complete. In the following we give a formulation for complete graphs.

Theorem 3.3.5 *Let $G = (V, E)$ be the complete graph with node set V and let $E' \subset E$ be an edge set such that $(V, E \setminus E')$ is a complete rectangular $h \times 2$ grid graph with $h \geq 3$. Let $\mathcal{N} = \{T_1, T_2\}$ be the net list, where $T_1 = \{(1,1),(h,2)\}$ and $T_2 = \{(1,2),(h,1)\}$. Set $F = E'$ and let $F_1, F_2 \subset E \setminus F$. Finally, set $\bar{k} = 3 - k$ for $k = 1, 2$. Then, the inequality*

$$(\chi^{E\setminus(F\cup F_1)}, \chi^{E\setminus(F\cup F_2)})^T x \geq 1$$

is valid for $P_{STP}(G, \mathcal{N}, \mathbb{1})$ if and only if F_1 and F_2 satisfy the following properties (see Figure 3.7):

(i) $F_k \subseteq \mathcal{F}_k := \{[(i,\bar{k}),(i+1,k)] : i = 1, \ldots, h-1\}$ for $k = 1, 2$.

(ii) For all $[(i_k,\bar{k}),(i_k+1,k)] \in F_k$, $k = 1, 2$ holds $i_1 \neq i_2$.

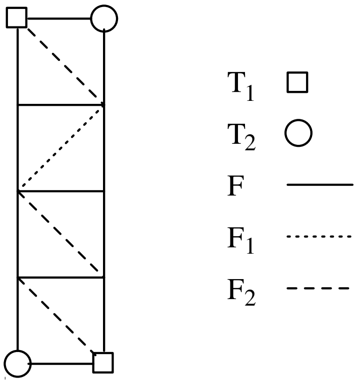


Figure 3.7: Illustration of a 5×2 grid inequality in a complete graph

Proof. First, we prove that (i) and (ii) are sufficient. Let $P = (S_1, S_2)$ be an arbitrary Steiner tree packing. W. l. o. g. S_1 and S_2 are paths. Suppose that $a^T\chi^P = 0$. For the same reason as in the proof of Theorem 3.3.4 there does not exist a Steiner tree packing in (V, F). This implies that $(S_1 \cap F_1) \cup (S_2 \cap F_2) \neq \emptyset$. Let $[(i_k,\bar{k}),(i_k+1,k)] \in (S_1\cap F_1)\cup(S_2\cap F_2)$ such that i_k is minimal. We consider the case $k = 1$ (the case $k = 2$ can be shown analogously). Obviously, $J_k \subset S_k$ for $k = 1, 2$, where $J_k = \{[(i,k),(i+1,k)] : i = 1, \ldots, i_1 - 1\}$. Since $[(i_k,\bar{k}),(i_k+1,k)] \in S_1$ and S_1 is a path, we obtain that either $[(i_1,1),(i_1,2)], [(i_1+1,1),(i_1+1,2)] \in S_1$ or $[(i_1,1),(i_1+1,1)], [(i_1,2),(i_1+1,2)] \in S_1$. In the first case set $W = \{(i,j) : i = 1, \ldots, i_1 - 1, j = 1, 2\} \cup \bigcup_{i\in I}\{[(i,2),(i+1,2)]\}$, where $I = \{i \in \{i_1, \ldots, h\} : [(i,2),(i+1,1)] \in F_1 \cap S_1$ and $[(i',2),(i'+1,1)] \in F_1 \cap S_1$ for all $i' = i_1, \ldots, i-1\}$. In the second case set $W = \{(i,j) : i = 1, \ldots, i_1, j = 1, 2\}$. Properties (i) and (ii) imply that $(\delta(W) \cap (F \cup F_2)) \setminus S_1 = \emptyset$. Since $(1,2) \in W$ and $(h,1) \in V \setminus W$, it follows that $(a^2)^T\chi^{S_2} \geq 1$, a contradiction.

It can be checked that the two conditions (i) and (ii) are also necessary. $\square$

If we add the condition "F_1 and F_2 are maximal with respect to (i) and (ii)" in Theorem 3.3.5, we obtain that these three conditions are necessary and sufficient for the grid inequality to be facet-defining (Martin [1992]).

3.3.3 Critical Cut Inequalities

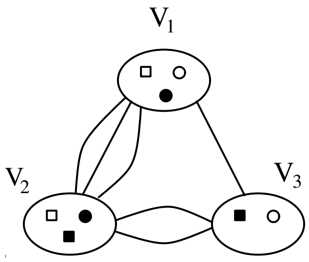
Definition 3.3.6 *Let $G = (V, E)$ be a graph with edge capacities $c_e \in \mathbb{N}$, $e \in E$. Moreover, let $\mathcal{N} = \{T_1, \ldots, T_N\}$ be a net list. For a node set $W \subseteq V$ we define $S(W) := \{k \in \{1, \ldots, N\} : T_k \cap W \neq \emptyset,\ T_k \cap (V \setminus W) \neq \emptyset\}$.*

(a) We call a cut induced by a node set W critical for STP $(G, \mathcal{N}, c)$, *if $s(W) := c(\delta(W)) - |S(W)| \leq 1$.*

(b) If V_1, V_2, V_3 is a partition of V such that $\delta(V_1)$ is a critical cut and if $T_1 \cap V_1 = \emptyset$ and $T_1 \cap V_i \neq \emptyset$ for $i = 2, 3$, we call the inequality

$$x^1([V_2 : V_3]) \geq 1$$

a critical cut inequality with respect to T_1. *(See Figure 3.8.)*



Consider the partition V_1, V_2, V_3 in Figure 3.8. Suppose the capacities of the edges are equal to one. Then, $\delta(V_1)$ is a critical cut. The critical cut inequality says that the net depicted by black rectangles must use at least one of the edges of $[V_2 : V_3]$.

Figure 3.8: Illustration of a critical cut inequality

The critical cut inequality is valid for $P_{\mathrm{STP}}(G, \mathcal{N}, c)$ for, suppose not, there exists a Steiner tree packing $(S_1, \ldots, S_N)$ with $|S_1 \cap \delta(V_1)| \geq 2$. This implies that $0 \leq c(\delta(V_1) \setminus S_1) - |S(V_1)| \leq c(\delta(V_1)) - 2 - |S(V_1)| \leq -1$, since $1 \notin S(V_1)$ and $\delta(V_1)$ is critical, a contradicition.

It turns out that under certain conditions this inequality is also facet-defining. Such details are reported in Martin [1992] and Grötschel, Martin, and Weismantel [1995].

All inequalities described are used in our branch-and-cut algorithm that we describe in the next section. The interested reader will find further classes of facet-defining inequalities in Grötschel, Martin, and Weismantel [1996c].

3.4 Algorithmic Aspects

In this section we discuss the separation problem of the classes of inequalities presented in the last two sections and describe a primal heuristic. Moreover, we point to several implementation aspects that suplement the general branch-and-cut algorithm described in Appendix B.

3.4.1 Separation Algorithms

We present in this subsection the main ideas of our separation algoirthms. The complete separation algorithms and the associated correctness proofs are quite complicated. For more information on this issue we refer the reader to Martin [1992] and Grötschel, Martin, and Weismantel [1996d].

Separation of the Steiner Partition Inequalities

Due to Theorem 3.2.3 every facet-defining inequality for the Steiner tree polyhedron yields a valid and, in case G is complete and the net list is disjoint, a facet-defining inequality for $P_{\mathrm{STP}}(G, \mathcal{N}, c)$. We focus here on one class of facet-defining inequalities that was characterized in Grötschel and Monma [1990] and generalizes the Steiner cut inequalities (3.1) (i). Let G be a graph and $T \subseteq V$ be a terminal set. We call a partition $V_1, \ldots, V_p$, $p \geq 2$, of V a *Steiner partition with respect to T*, if $V_i \cap T \neq \emptyset$ for $i = 1, \ldots, p$. The inequality

$$x(\delta(V_1, \ldots, V_p)) \geq p - 1$$

is called *Steiner partition inequality.* It is valid for $P_{\mathrm{STP}}(G, \{T\}, \mathbb{1})$ and Grötschel and Monma have characterized conditions under which it defines a facet. Though the corresponding separation problem is $\mathcal{NP}$-complete in general (see Grötschel, Monma, and Stoer [1992]), there exist special cases for which it can be solved in polynomial time. One of these special cases is obtained if we restrict the graph G to be planar and the set of terminal nodes T to lie on the outer face of G. This special case is of particular practical interest, because it includes the switchbox routing problem. The main idea of the algorithm for solving the separation problem in this case is as follows.

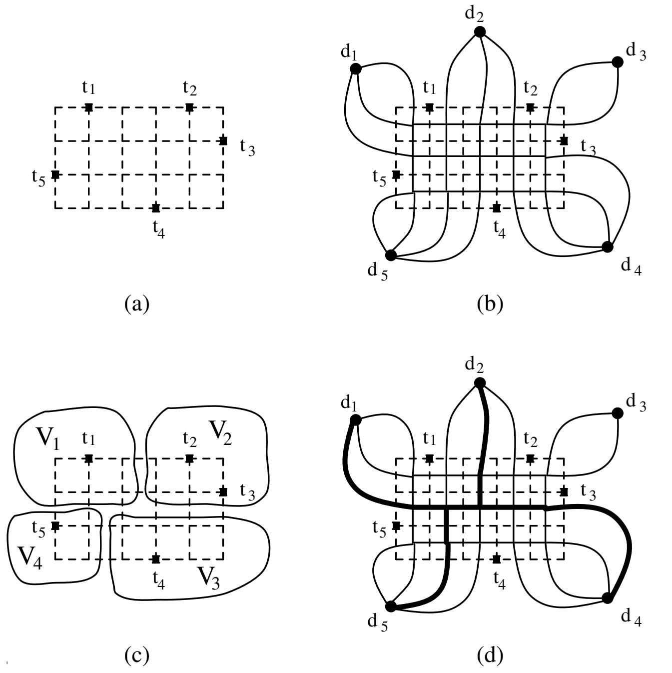


Figure 3.9: Separation of Steiner partition inequalities

Without loss of generality we can assume that G is 2-node connected (otherwise the graph can be decomposed). Thus, the edge set that encloses the outer face of G is a cycle. Suppose the terminal set $T = \{t_1, \ldots, t_z\}$ is numbered in a clockwise fashion along this cycle. Now, consider the dual graph $G^* = (V^*, E)$ of G and subdivide the node representing the outer face in z nodes $d_1, \ldots, d_z$ such that every edge belonging to a path in G from t_i to t_{i+1} on the outer face is now incident to d_{i+1} for $i = 1, \ldots, z$. Let $G_D = (V_D, E)$ denote the resulting graph and set $D = \{d_1, \ldots, d_z\}$ (cf. Figure 3.9 (a) and (b)).

It turns out that under mild assumptions every edge set $S = \delta(V_1, \ldots, V_i)$ induced by a Steiner partition $V_1, \ldots, V_i$ is in one-to-one correspondence with an edge-minimal Steiner tree in G_D with respect to some subset $J \subseteq D$ (cf. Figure 3.9 (c) and (d)).

This equivalence yields that the problem of separating the class of Steiner partition inequalities reduces to the problem of finding a subset J of D and an edge-minimal Steiner tree in G_D with respect to J. Given J, we can determine an optimal Steiner tree in G_D with respect to J by applying the dynamic programming approach proposed in Dreyfus and Wagner [1971] and Erickson, Monma, and Veinott [1987]. Thus, the crucial point is to find the subset $J \subseteq D$. In Grötschel, Martin, and Weismantel [1996d] we show that we can locally decide which terminal belongs to an optimal solution. This observation can be taken into account by modifying the recursion formula of the dynamic program appropriately.

The algorithm for separating the Steiner partition inequalities gives rise to several heuristic procedures. Instead of calculating the optimal Steiner tree in G_D we heuristically determine Steiner trees. For more details, we again refer to our paper Grötschel, Martin, and Weismantel [1996d].

Separation of the Alternating Cycle Inequalities

Consider an instance STP $(G, \mathcal{N}, c)$ of a Steiner tree packing problem with $\mathcal{N} = \{T_1, T_2\}$. How to find suitable pairs T_1 and T_2 will be discussed in the next section. It is open whether the separation problem of alternating cycle inequalities for $P_{\mathrm{STP}}\,(G, \mathcal{N}, c)$ is solvable in polynomial time. We restrict our attention to the case where G is planar and all terminals lie on the outer face of G. Here, our idea to separate alternating cycle inequalities is to apply dynamic programming techniques in a similar way as was done for finding Steiner partition inequalities.

Again, we show that alternating cycle inequalities are in a one-to-one mapping with Steiner trees in an appropriate dual graph. In this case, however, these Steiner trees have to satisfy many technical conditions.

In particular, these technical conditions cause that some edges are evaluated differently for different nets. This is due to the fact that for the alternating cycle inequality, edge sets F (edges which have a zero coefficient for both nets), F_1 (edges which have a zero coefficient just for net 1) and F_2 (edges which have a zero coefficient just for net 2) are involved (cf. Theorem 3.3.2). Unfortunately, taking all these constraints into account we obtain a dynamic program, whose optimum solution does not necessarily correspond to the most violated alternating cycle inequality. Rather, the optimum value found by the dynamic program provides just a lower bound for the most violated alternating cycle inequality. If this value is non-negative, we can guarantee that there does not exist a violated inequality of this type. Otherwise, there may exist a violated alternating cycle inequality, but the algorithm terminates with an edge set that does not correspond to an alternating cycle inequality (see Grötschel, Martin, and Weismantel [1996d]).

The relationship between alternating cycle inequalities and Steiner trees satisfying certain technical conditions in the appropriate dual graph gives rise to many heuristics. Again, we have implemented, in addition to the dynamic program, an algorithm that determines heuristically such Steiner trees and checks whether the corresponding alternating cycle inequalities are violated.

Finding Critical Cuts

Remember that a cut induced by a set of nodes W is critical, if $s(W) = c(\delta(W)) - |S(W)| \leq 1$, where $S(W) = \{k \in \{1, \ldots, N\} : T_k \cap W \neq \emptyset,\ T_k \cap (V \setminus W) \neq \emptyset\}$. In the following we briefly explain why we concentrate on the problem of finding critical cuts rather than on the separation problem for the critical cut inequalities itself.

First, let us point out that, from a practical point of view, we are interested in Steiner tree packings where each of the single Steiner trees is edge-minimal. Since a positive objective function is minimized, we know in advance that the weight-minimal Steiner trees are also edge-minimal, and we exploit this property to reduce the problem size.

Suppose $W \subseteq V$ is a node set and T_k is a set of terminals with $T_k \subseteq W$ or $T_k \subseteq V \setminus W$. Then any edge-minimal Steiner tree for T_k that uses one edge of $\delta(W)$ has to contain at least two of these edges. But, if $\delta(W)$ is a critical cut then at most one edge of $\delta(W)$ can be used by the Steiner tree for T_k. Hence, the following variables can be fixed accordingly, i. e.,

$$\begin{aligned} x_e^k = 0, &\quad \text{for all } k \in \{1, \dots, N\} \setminus S(W),\, T_k \subseteq W,\, e \in E(V \setminus W) \cup \delta(W); \\ x_e^k = 0, &\quad \text{for all } k \in \{1, \dots, N\} \setminus S(W),\, T_k \subseteq V \setminus W,\, e \in E(W) \cup \delta(W). \end{aligned}$$

Let us now point out the relationship to the critical cut inequality. Consider the situation in Definition 3.3.6 (b), where V_1, V_2, V_3 is a partition of V such that $\delta(V_1)$ is a critical cut and $T_1 \cap V_1 = \emptyset$ and $T_1 \cap V_i \neq \emptyset$, $i = 2, 3$. Since $\delta(V_1)$ is critical, we can fix all variables x_e^1 to zero for $e \in \delta(V_1)$. Thus, by fixing these variables we can separate the critical cut inequalities via separating the Steiner cut inequalities. For example, a Steiner cut inequality for T_1 of the instance described in Definition 3.3.6 (b) is $x^1(\delta(V_2)) = x^1([V_2 : V_1]) + x^1([V_2 : V_3]) \geq 1$. By taking the fixed variables into account we obtain the critical cut inequality $x^1([V_2 : V_3]) \geq 1$.

In the remainder of this subsection we briefly sketch the ideas how to find critical cuts. We restrict ourselves to instances STP $(G, \mathcal{N}, \mathbb{1})$, where G is a complete rectangular grid graph and all terminal sets of the net list $\mathcal{N}$ lie on the outer face of G. Here, we can show (Grötschel, Martin, and Weismantel [1995]) that, if there exists a node set $W \subset V$, $W \neq \emptyset$ that induces a critical cut, there exists

(i) a node $w \in V$ such that $\delta(w)$ is a critical cut with respect to STP $(G, \mathcal{N}, \mathbb{1})$ or

(ii) a horizontal or vertical critical cut with respect to STP $(G, \mathcal{N}, \mathbb{1})$. (A cut F is called *horizontal* if there exists some $i \in \{1, \dots, h-1\}$ such that $F = \{uv \in E : u = (i, j) \text{ and } v = (i+1, j) \text{ for some } j \in \{1, \dots, b\}\}$; a vertical cut is defined analogously).

Based on this observation we can now develop an algorithm for finding critical cuts. We check, for all nodes $v \in V$, whether $\delta(v)$ is critical. In addition, we also check whether there exist critical vertical or horizontal cuts. If we do not succeed in finding a critical cut, we can conclude that none exists. Otherwise, we fix the corresponding variables. In order to find further critical cuts, we inductively enlarge the node set $W = \{v\}$ in all four possible directions of the grid in a greedy like fashion. The variables of the critical cuts found this way are fixed accordingly.

Finally, we have developed a heuristic for separating grid inequalities that proceeds in a greedy-like fashion. Details can be found in Grötschel, Martin, and Weismantel [1995].

3.4.2 A Primal Heuristic

This section is devoted to describing our primal heuristic. The idea of the heuristic is to make use of the information given by the actual solution of the cutting plane phase.

We have developed a sequential algorithm. We consider each terminal of a net to be an (isolated) component. We iteratively connect two components of a net according to an a-priori determined sequence. However, we do not apply this scheme by routing one net completely after another, but we connect only two components in each iteration. The success of such a procedure strongly depends on the predefined sequence. In our algorithm this sequence is mainly determined by the solution $\bar{x}$ of the actual linear program. More precisely, we define a function f depending on $\bar{x}$ according to which the subsequent two components are selected. (A detailed explanation of the function f is given below.) We try to connect the two selected components via a shortest path. Since in a complete rectangular grid graph a shortest path is not unique in general, we have implemented further criteria according to which the choice is made. Besides others, these criteria depend on the location of the terminals of the other nets, the position of the not yet connected terminals of the same net and, again, on the solution $\bar{x}$. For a detailed description of these criteria we refer the reader to Martin [1992]. If it is possible to connect the two components on a shortest path by taking the mentioned criterion into account, we connect these two components and choose the next pair of components. Otherwise, we recompute the function f and the sequence by taking the already connected components into account. This iterative procedure is continued until all nets are connected or no further components can be connected.

The crucial point for the success of the described heuristic is the choice of the function f. For the definition of the function f let us assume that the nodes be numbered such that $V = \{(i,j) : i = 1,\dots,h,\ j = 1,\dots,b\}$ and let $V_{l,r,t,d} := \{(i,j) : i = l,\dots,r,\ j = t,\dots,d\}$ for $l,r \in \{1,\dots,b\}, l < r$ and $t,d \in \{1,\dots,h\}, t < d$. Consider some net $k \in \{1,\dots,N\}$. Let S_k be the edge set that was already determined for connecting T_k, T'_k the set of not yet connected terminals and $\hat{G} = (V, \hat{E})$ the graph that is obtained from the given complete rectangular grid graph by deleting all edges that are already used for the connection of terminals.

We consider the case $S_k \neq \emptyset$ (in the case $S_k = \emptyset$ the function is defined similarly) and let $s_k = (i_s, j_s) \in T'_k$ and $t_k = (i_t, j_t) \in V(S_k)$ be given. Determine $l, r \in \{1,\dots,b\}, l < r$ and $t,d \in \{1,\dots,h\}, t < d$ such that $s_k, V(S_k) \in V_{l,r,t,d}$ and $|V_{l,r,t,d}|$ is minimal. Set $E_{l,r,t,d} = \{e \in \hat{E}(V_{l,r,t,d}) : \bar{x}^k_e > 0\}$ and suppose (V_s, E_s) is the component in $(V_{l,r,t,d}, E_{l,r,t,d})$ with $s_k \in V_s$. Set

$$f_{\bar{x}^k}(s_k, t_k) := |w\big(W(s_k,t_k)\big) - \sum_{e \in E_s} w_e\, \bar{x}^k_e|, \tag{3.3}$$

where $W(s_k, t_k)$ is a shortest path from s_k to t_k in $\hat{G}$ (with respect to w). We choose those two components for being connected next that minimize this function f.

The heuristic idea of this function is the following. We determine a graph $(V_{l,r,t,d}, E_{l,r,t,d})$ which is the smallest rectangular grid graph containing both components (often designated as the "minimal enclosing rectangle"). Inside the minimal enclosing rectangle we compute the weighted sum $(= \omega)$ of those edges that are in the same component as s_k, where only edges with $\bar{x}^k_e > 0$ are considered. The value ω is compared to the length $(= \lambda)$ of a shortest path between the two nodes. If ω is smaller than λ, we assume that the information from $\bar{x}^k$ is too poor to decide how to connect the two nodes. The smaller the difference, the less information and the greater the value of f. On the other hand, if ω is greater than λ the two nodes will be probably connected via a detour. The greater the difference, the greater the value of f. Thus, we choose the components with value ω close to λ first.

Obviously all ideas mentioned so far are of heuristic nature and there is no guarantee that we will obtain good results. However, due to many tests we have performed this strategy seems to be reasonable.

3.4.3 Further Issues

We tuned the general branch-and-cut algorithm described in Appendix B to solve practical Steiner tree packing problems. We briefly discuss some of the features in this section. Further details on the implementation may be found in Grötschel, Martin, and Weismantel [1996b].

One major problem we encountered is that the linear programs appeared to be quite difficult. One of the reasons for this is probably that our linear programs have many alternative optimum solutions and are simultaneously primally and dually highly degenerate. Even CPLEX, a fast and robust code for solving linear programs, had enormous difficulties to solve the linear programs.

A frequently used method to overcome such difficulties is to perturb the right-hand side of the linear program. Since we are solving the problems with the dual simplex method we must perturb the objective function of the weighted Steiner tree packing problem. After many experiments and discussions with R.E. Bixby (Rice University and ILOG) we decided to proceed as follows. Let $\omega \in \mathbb{R}^{\mathcal{N}\times E}$ with $\omega_e^k = w_e$ for all $e \in E$, $k = 1, \ldots, N$ be the original objective function. For each terminal set T_k, we compute a Steiner tree S_k by applying a heuristic procedure and determine random numbers $\varepsilon_e^k \in [0,1]$. Then we use the objective function vector $\widetilde{w} \in \mathbb{R}^{\mathcal{N}\times E}$ defined by

$$\begin{aligned} \widetilde{w}_e^k &:= \omega_e^k - b\,\varepsilon_e^k - \eta, && \text{if } e \in S_k, \text{ for } k = 1, \ldots, N; \\ \widetilde{w}_e^k &:= \omega_e^k - b\,\varepsilon_e^k, && \text{if } e \notin S_k, \text{ for } k = 1, \ldots, N, \end{aligned}$$

where $\eta = \frac{1}{2(n+1)}$ and $b = \min\{10^{-5}, \frac{1}{2\,(n+1)}\}$ in the actual implementation. It is easy to see that, if the given objective function is integer, an optimal Steiner tree packing with respect to $\widetilde{w}$ is also optimal with respect to ω and vice versa. The success of the perturbation trick is very impressive, the running times can be reduced by a factor of ten, see Grötschel, Martin, and Weismantel [1996b].

Another (polyhedral) preprocessing trick helped to increase the lower bounds and to decrease the running time considerably. After "solving" the trivial initial linear program by setting all variables to zero we do not call our general separation routines; rather, we generate a particular class of Steiner cut and Steiner partition inequalities for which we have heuristic reasons to believe that they form a sensible set of "good" initial cutting planes.

Since the underlying graph is a complete rectangular grid graph, we add all Steiner cut inequalities that are induced by a horizontal or vertical cut. The advantage is that these inequalities have pairwise disjoint support. In addition, for multi-terminal nets we extend each Steiner cut inequality to a Steiner partition inequality with right-hand side greater than two. For example, let $|T_k| = p \geq 3$, $F = \delta(W)$, $W \subset V$, be a vertical cut that induces a Steiner cut inequality. First, we determine a Steiner partition $W_1, \ldots, W_q$ of W such that $[W_i : W_{i+1}]$ is a horizontal cut in $(W, E(W))$ for $i = 1, \ldots, q-1$ and q is maximal. The only node sets of $W_1, \ldots, W_q$ that possibly contain more than one terminal are W_1 and W_q. For these two node sets we again determine a Steiner partition $W_r^1, \ldots, W_r^{l_r}$ for $r = 1$ and $r = q$ such that $[W_r^i : W_r^{i+1}]$ is a vertical cut in $(W_r, E(W_r))$ and l_r is maximal. The same procedure is applied to the node set $V \setminus W$. Taking both together we obtain (after renumbering) a Steiner partition $W_1, \ldots, W_s$ with $s = p$, and $x(\delta(W_1, \ldots, W_s)) \geq p-1$ defines a Steiner partition inequality. We extend each horizontal and vertical cut that defines a Steiner cut inequality in this way. Obviously, the resulting inequalities do not necessarily have disjoint support, but the right-hand side is quite large. Let us denote all inequalities constructed in this way and the Steiner cut inequalities induced by a horizontal or vertical cut by *special Steiner partition inequalities.*

Iter.	with special Steiner part. ineq.		without special Steiner part. ineq.	
	LP value	CPU time	LP value	CPU time
1	0.00	0:02	0.00	0:02
2	456.56	0:05	196.87	0:05
3	457.57	0:11	351.13	0:11
4	457.59	0:22	374.20	0:27
5	457.60	0:38	389.38	1:12
10	459.00	6:18	425.25	7:54
15	460.66	17:36	440.42	22:03
20	461.70	33:33	447.07	43:21

Table 3.1: Progress by using special Steiner partition inequalities

Table 3.1 illustrates the progress we obtain for example difficult switchbox by using the special Steiner partition inequalities after solving the initial linear program. Column 1 presents the number of cutting plane iterations. Column 2 and 3 (resp. 4 and 5) give the LP objective value and the accumulated CPU-time (in min:sec) by using (resp. not using) the special Steiner partition inequalities after the first iteration. The results are impressive. The lower bound we obtain within five seconds after the second iteration by adding the special Steiner partition inequalities is much better than after running the algorithm with the separation algorithms for the Steiner partition inequalities discussed above for more than forty minutes.

Next, we want to deal with the separation of the alternating cycle inequalities. The separation algorithms we have outlined (the dynamic program as well as the heuristics) need a pair of nets as input. The problem we are concerned with is to choose one (or several) "good" pairs of terminal sets for which we want to execute the separation algorithms. If we would call one of these algorithms for all net pairs, we would obtain a non-acceptable running time, because the number of calls is quadratic in the number of nets. Note that we encountered a similar problem for the separation of multiple cover inequalities, see page 17.

In order to overcome this problem here, we try to exploit the information given by the primal heuristic. Remember that two components are gradually connected via a shortest path. If this is not possible, another net must block this path. Obviously the two nets concurrently prefer certain edges in this case. Moreover, this situation indicates that the information provided by the linear programming solution is too poor to decide which of the nets is forced to make a detour, see the definition of the function f in (3.3). Hence, we conclude that more inequalities combining these nets are necessary. Thus, we call the separation algorithms for the alternating cycle inequalities for nets that are in conflict due to the information of the primal heuristic. Practical experiments have shown that the number of such conflicts is sublinear in the number of nets and that strongly violated alternating cycle inequalities can be obtained for such conflicting net pairs.

We want to point out that not only the linear program solution supplies important information for the primal heuristic. But also conversely, the primal heuristic indicates which type of inequalities are promising for a further execution of the cutting plane algorithm. In our opinion this interplay of the methods for determing the lower and upper bound is essential in order to solve large scale problems.

3.5 Computational Results

In this section we report on computational experiences with our branch-and-cut algorithm. We have tested the algorithm on switchbox routing problems that are discussed in the literature. Table 3.2 summarizes the data. Column 1 presents the name used in the literature. In Column 2 and 3 the height and width of the underlying grid graph is given. Column 4 contains the number of nets. Columns 5 to 9 provide information about the distribution of the nets; more precisely, Column 5 gives the number of 2-terminal nets, Column 6 gives the number of 3-terminal nets and so on. Finally, the footnotes state the references to the papers the examples are taken from.

Example	h	w	N	Distribution of the Nets				
				2	3	4	5	6
difficult switchbox[a]	15	23	24	15	3	4	1	1
more difficult switchbox[b]	15	22	24	15	3	5	0	1
terminal intensive switchbox[c]	16	23	24	8	7	5	4	0
dense switchbox[c]	17	15	19	3	11	5	0	0
augmented dense switchbox[c]	18	16	19	3	11	5	0	0
modified dense switchbox[b]	17	16	19	3	11	5	0	0
pedagogical switchbox[b]	16	15	22	14	4	4	0	0

[a]Burstein and Pelavin [1983]
[b]Cohoon and Heck [1988]
[c]Luk [1985]

Table 3.2: Switchbox routing problems: Data

In all examples as they were originally introduced in the literature, the underlying graph is given as follows. The graph is obtained from a complete rectangular grid graph by removing the outer cycle, see Figure 3.10 (a). Hence, every terminal is incident to a unique edge, and obviously, every Steiner tree must contain this edge. It is easy to see that by contracting all pending edges an equivalent problem is obtained, see Figure 3.10 (b). The graph resulting this way is a complete rectangular grid graph with terminals on the outer face. This instance is the input to our algorithm.

The first example difficult switchbox was introduced by Burstein and Pelavin. The second one more difficult switchbox is derived from the first one by deleting the last column. (More precisely, the edges $[(i,23),(i,24)]$ of the first grid graph are contracted for $i = 1,\dots,15$ and parallel edges are deleted.) The net list is the same. The difference in the distribution occurs (see Column 7 and 8), because an edge whose endpoints belong to the same net is contracted. The third problem instance was introduced by Luk, here each outer face node is occupied by a terminal. The fourth switchbox routing problem is again due to Luk. Up to now it was not known whether a solution for this example exists, if the Manhattan or 2-layer model is used. Based on this example two variants can be obtained. One, called augmented dense switchbox, has an additional column on the right, the other, called modified dense switchbox, has an additional column near the middle and an additional row on the buttom. The last example was introduced by Cohoon and Heck. They illustrated their algorithm on this problem.

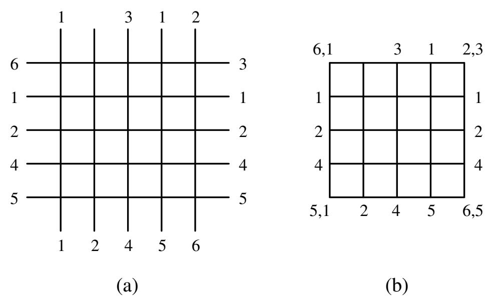


Figure 3.10: Deletion of preassigned edges

In all examples the edge weights as well as the edge capacities are equal to one. Unfortunately, the problem instances do not fix the routing model (Manhattan, knock-knee or multiple layer model). As outlined in the introduction the weighted Steiner tree packing problem reflects the knock-knee model, and thus the results of our algorithm apply to this case. A comparison of our results to results for the Manhattan model (and 2-layer model) will be given in the next section.

The data in Table 3.2 gives rise to integer programming formulations with a number of variables ranging from 9082 to 16728. As outlined in the last subsection of Section 3.4 certain variables can be fixed a-priori. The number of variables fixed this way ranges from 14% to 53%, resulting in a final number of variables between 4251 (for dense switchbox) and 13424 (for difficult switchbox).

In Table 3.3 the results we have obtained with our branch-and-cut algorithm are summarized. Column 2 gives the best feasible solution, and Column 3 the best lower bound. In Column 4 the percentage deviation of the best solution from the best lower bound is given; more precisely, Column 4 contains the value $\frac{w_2 - w_3}{w_3}$, where w_2 (resp. w_3) is the corresponding value of Column 2 (resp. 3). Column 5 (resp. 6) gives the number of cutting plane iterations, i. e., the number of solved linear programs, (resp. the number of nodes in the branching tree). Finally, the last column reports on the running times. The values are stated in minutes on a Sun Enterprise 3000 with a 168 MHz UltraSPARC processor. The two examples dense switchbox and augmented dense switchbox marked with an asterix are stopped after two days of computation, because no further progress could be achieved.

The numbers in Table 3.3 are quite encouraging. We solve 5 out of 7 problems to optimality, and for the remaining two problem instances the lower bound in Column 3 guarantees that the best feasible solution deviates at most 0.5% from the optimal solution. In our opinion the main advantage of our algorithm is that the quality of an heuristically determined solution can be evaluated with the lower bound. Especially, for problem instances arising in VLSI-Design, where in general only heuristics are at hand, a cutting plane algorithm helps in analyzing the heuristics and simultanously delivers a lot of knowledge about the problem itself.

The distribution of the cuts found is around the same for all problems, see Table 3.4. We find individual (i. e., Steiner cut- and Steiner partition-) inequalities in the thousands (between 3075 for modified dense switchbox and 23098 for dense switchbox), around a factor of ten less alternating cycle inequalities (between 298 for modified dense switchbox and 1851 for dense switchbox), and again around a factor

Example	Best Sol	Lower Bd	Gap	LPs	B&B	CPU-Time
difficult switchbox	464	464	0.0%	62	1	190:34
more difficult switchbox	452	452	0.0%	53	1	118:10
terminal intensive switchbox	536	536	0.0%	895	17	2226:46
dense switchbox*	441	439	0.5%	3420	27	2738:21
augmented dense switchbox*	469	467	0.4%	361	1	2863:51
modified dense switchbox	452	452	0.0%	58	1	42:22
pedagogical switchbox	331	331	0.0%	84	3	27:22

Table 3.3: Results for the knock-knee model

of ten less $h \times 2$ grid inequalities (at most 112 for terminal intensive switchbox). Thus, as in the case of the multiple knapsack problem one might ask the question whether joint inequalities help to solve the problems (faster). Indeed, they do. We tested on the five instances we solve to optimality the influence when the separation of joint inequalites is turned off. The results are much worse. We are not able to solve terminal intensive switchbox within the time limit of 60 hours. Also pedagogical switchbox is a disaster, we need 1817 branch-and-bound nodes to solve the problem, whereas with joint inequalities only three are necessary. These numbers clearly show the need of joint inequalities. The results are comfirmed by Grötschel, Martin, and Weismantel [1995], where additonal separation strategies have been tested, and it is always the best to separate all joint inequalites.

Example	Steiner partition	alternating cycles	$h \times 2$ grid
difficult switchbox	5236	747	16
more difficult switchbox	4383	556	18
terminal intensive switchbox	15234	1072	112
dense switchbox*	23098	1851	21
augmented dense switchbox*	11152	1083	18
modified dense switchbox	3075	298	23
pedagogical switchbox	3136	546	32

Table 3.4: Added cutting planes for the knock-knee model

Nevertheless, even with the separation of joint inequalities one major problem with our algorithm is its running time, where at least 90% of the total time for each problem is spent in the solution of the linear programs. The numbers in the last column of Table 3.3 are very high. One reason is that we are interested in an optimal

Example	Best Sol	Lower Bd	Gap	LPs	B&B	CPU-Time
difficult switchbox	464	464	0.0%	72	1	253:58
more difficult switchbox	452	452	0.0%	58	1	152:46
terminal intensive switchbox	539	535	0.0%	561	7	3592:52
modified dense switchbox	452	452	0.0%	427	1	2904:30
pedagogical switchbox	331	331	0.0%	1933	1817	1301:09

Table 3.5: Results for the knock-knee model without joint inequalites

solution or at least in the best lower and upper bound for each of the problems that we can achieve with our approach. This is time consuming. In practice, heuristics usually find feasible solutions for these instances in a few seconds. These running times are certainly not reachable with our algorithm. However, the main advantage of the cutting plane approach is to give a solution guarantee for the best known feasible solution. We are not aware of any method used in practice that is able to guarantee a certain quality of the feasible solutions found. From this point of view, we have analyzed our results also. It turns out that, though the running time behaviour is exponential, for all problem instances, the lower bound deviates from the best feasible solution only by at most 5% percent after no more than a minute.

In our opinion these times are acceptable. However, we would like to point out that these examples are quite small in comparison to problem sizes arising in other practical applications for the design of electronic circuits. It is definitely a long-term goal to apply branch-and-cut algorithms to instances of larger scale.

Comparing Routing Models

From a practical point of view a very interesting question with probably never ending discussions is the question which routing model should be preferred: the knock-knee model, the Manhattan model or the multiple layer model. To our knowledge all methods from the literature use the Manhattan model or the 2-layer model. The choice of the underlying model strongly influences the solvability of the problems. The theory says that in the knock-knee model two layers may not suffice, whereas in the Manhattan model (and the 2-layer model) they do. Moreover, there are examples where there exists a solution in the 2-layer model, whereas it does not in the knock-knee model. Figure 3.10 illustrates such an example (this example is taken from Cohoon and Heck [1988]). On the other hand, one can expect that the wiring length that is needed when Steiner trees are packed in the knock-knee model is smaller than in case of the Manhattan model. But, does the knock-knee model provide substantially shorter wiring lengths? We have tried to answer these questions for the problem instances introduced in the last section.

To model the Manhattan routing style, where knock-knees are not allowed, we have to introduce additional inequalities that make it impossible for two Steiner trees to bend at the same node, see Figure 3.3.

Let $\mathrm{STP}_w\,(G,\mathcal{N},\mathbb{1},w)$ be an instance of the (weighted) Steiner tree packing problem, where G is a complete rectangular grid graph and uv, vw are two consecutive horizontal (or vertical) edges. Let N_1, N_2 be a partition of $\{1,\dots,N\}$. Then, the constraint

$$\sum_{k\in N_1} x^k_{uv} + \sum_{k\in N_2} x^k_{vw} \leq 1 \tag{3.4}$$

is called *Manhattan inequality.*

It is easy to see that every edge-minimal packing of Steiner trees that satisfies, for every pair of consecutive edges and for every 2-partition of the set of nets, the corresponding Manhattan inequality (3.4) and the constraints (3.1) (i) – (iv) corresponds to a feasible switchbox routing in the Manhattan style, and vice versa.

We extended and modified our branch-and-cut algorithm to handle switchbox routing in the Manhattan style as well. All separation routines and all special features (preprocessing, ...) and implementation tricks (perturbation, ...) discussed so far can be taken over. In addition, we designed and implemented a separation routine for the Manhattan inequalities (3.4), and some (minor) changes were needed or useful to apply the code to Manhattan routing problems that we briefly discuss now.

Our procedure for separating Manhattan inequalities works as follows. Let us assume that the capacity inequalities are satisfied by the current LP solution $\bar{x}$ (of course, this can be checked in linear time). Let $uv \in E$ and $vw \in E$ be two horizontal edges that are incident to node $v \in V$ (the same arguments apply to the case of two consecutive vertical edges). For every net $k \in \{1,\ldots,N\}$, we determine $\max\{\bar{x}^k_{uv}, \bar{x}^k_{vw}\}$. Set $N_1 := \{k \in \{1,\ldots,N\} : \bar{x}^k_{uv} > \bar{x}^k_{vw}\}$ and $N_2 := \{k \in \{1,\ldots,N\} : \bar{x}^k_{uv} \leq \bar{x}^k_{vw}\}$. If $N_1 = \emptyset$ or $N_2 = \emptyset$, we can conclude that no violated Manhattan inequality exists, since the capacity inequalities are all satisfied. Otherwise, N_1, N_2 is a partition of $\{1,\ldots,N\}$ and the inequality $\sum_{k\in N_1} x^k_{uv} + \sum_{k\in N_2} x^k_{vw} \leq 1$ is a Manhattan inequality with maximal left-hand side. This procedure obviously solves the separation problem for the class of Manhattan inequalities.

We also modified the LP-based primal heuristic to guarantee that only Steiner tree packings are feasible that contain no knock-knees. We omit the technical details here. Finally, we exploit the fact that nets must not bend against each other in order to fix variables at the initial phase of the code, for details see Grötschel, Martin, and Weismantel [1997].

Table 3.6 summarizes our results. For all instances we could either find an optimal solution or prove that the problem is infeasible. The latter situation occurred in the two cases dense switchbox and modified dense switchbox. To our knowledge, it was up to now open whether there exists a packing of Steiner trees in the Manhattan model for these instances. Actually, the two examples modified dense switchbox and augmented dense switchbox are extensions of the problem dense switchbox in which additional tracks are added (augmented dense switchbox has an additional vertical track on the right and modified dense switchbox has an additional vertical track near the middle and an additional horizontal track at the bottom). In fact, these modifications had been introduced because no routing algorithm could find a feasible solution for dense switchbox in any routing style. Whereas a Manhattan routing is known for the problem augmented dense switchbox, the heuristics described in the literature were unable to find one for modified dense switchbox. Our algorithm yields a mathematical proof that, indeed, no routing routine can ever be successful for the latter example.

Table 3.7 presents a comparison of the lower and upper bounds obtained with our algorithm in the knock-knee and Manhattan style. The results are quite different for different instances. For one example the wiring length in the Manhattan model

Example	Best Sol.	Lower Bd	Gap	LPs	B&B	CPU-Time
difficult switchbox	469	469	0.0%	148	3	380:54
more difficult switchbox	461	461	0.0%	424	7	977:57
terminal intensive switchbox	537	537	0.0%	28	1	46:47
dense switchbox	-	∞	-	20	1	10:18
augmented dense switchbox	469	469	0.0%	30	1	53:30
modified dense switchbox	-	∞	-	47	1	75:28
pedagogical switchbox	343	343	0.0%	350	29	240:60

Table 3.6: Results for the Manhattan model

is just the same as in the knock-knee model though the solutions have indeed knock-knees. For four other problem instances the wiring length in the Manhattan model exceeds that in the knock-knee model by a small amount (for difficult switchbox by 5 (= 1.1%), for more difficult switchbox by 9 (= 2.0%), for terminal intensive switchbox by 1 (= 0.1%), and for pedagogical switchbox by 12 (= 3.6%)). Of course, the shorter lengths in the knock-knee model must be paid by additional layers. Since the percentage of increase in length is quite small one may tend to prefer the Manhattan model. However, for the examples dense switchbox and modified dense switchbox, for which we could prove that there does not exist a feasible solution in the Manhattan model, we are able to find feasible solutions in the knock-knee model. This makes the knock-knee model more attractive.

Example	Knock-knee Model		Manhattan Model	
	Lower Bd	Upper Bd	Lower Bd	Upper Bd
difficult switchbox	464	464	469	469
more difficult switchbox	452	452	461	461
terminal intensive switchbox	536	536	537	537
dense switchbox	438	441	∞	-
augmented dense switchbox	467	469	469	469
modified dense switchbox	452	452	∞	-
pedagogical switchbox	331	331	343	343

Table 3.7: Comparing the knock-knee and Manhattan model

Comparing running times we observe similar phenomena (see the last columns in Tables 3.3 and 3.6). Some examples are quite easy for the knock-knee model but rather hard for the Manhattan model, and vice versa, some are solved quite fast in the Manhattan model, but are difficult in the knock-knee style. Based on these results we cannot decide whether one model is superior to the other. The issue of choosing the "correct" model must be left to practitioners and depends on the chosen fabrication technology and the given design rules.

Finally, we have compared our results with those published in the literature. In Table 3.8 we summarize the objective function values of the – to our knowledge – best Manhattan solution reported in the literature (Column 2). No entry means that we did not find any Manhattan solution for the corresponding problem instance that was published in the literature. In Column 3 the objective function value of the Manhattan solution that was obtained by our code is shown. The values differ from those reported in Table 3.6 and Table 3.7, respectively, by the total number of terminals of the original data due to preprocessing (see page 45 for further explanations). For the examples dense switchbox and modified dense switchbox no Manhattan solution exists which is indicated by the symbol "$*$" in Column 3. For the problem instance augmented dense switchbox the solution given in Luk [1985] is optimal, whereas for the two problems difficult switchbox and terminal intensive switchbox the solution found by our code improves the best solution reported in the literature by 2.2% and 2.7%, respectively.

Example	Best Manhattan the Literature	Solution from our Code
difficult switchbox	547[a]	535
more difficult switchbox	-	527
terminal intensive switchbox	632[b]	615
dense switchbox	-	$*$
augmented dense switchbox	529[b]	529
modified dense switchbox	-	$*$
pedagogical switchbox	-	400

[a]Jou, Lee, Sun, and Wang [1990]
[b]Luk [1985]

Table 3.8: Best solutions for the Manhattan model

Of course, there are further routing algorithms presented in the VLSI literature. To our knowledge, all of them apply to the 2-layer model, see, for instance, Lin, Hsu, and Tsai [1988], Joobbani and Siewiorek [1986], Cohoon and Heck [1988], Jou, Lee, Sun, and Wang [1990], Gerez and Herrmann [1989], Tzeng and Séquin [1988]. A comparison of the knock-knee or Manhattan model to the 2-layer model is difficult. In the 2-layer model two different nets may run on the same horizontal or vertical edges of the two layers. The number of consecutive edges that are used on both layers is usually limited in order to avoid so-called cross-talk problems. The value of this upper bound depends on the design rules and technological constraints, but is mostly neglected by the routing algorithms.

The fact that the wires can run on top of each other along arbitrary lengths may lead to routings with shorter wiring lengths than in the Manhattan model, because a solution in the Manhattan model is feasible for the 2-layer model. Nevertheless, we have compared our Manhattan solutions to the best 2-layer solutions reported in the literature. It turns out that for all examples for which a Manhattan solution exists, the objective function values are at most 1% worse than the objective function values of the corresponding 2-layer solutions. In fact, for the two examples terminal intensive switchbox and augmented dense switchbox the Manhattan solution provides the same wiring length, and for more difficult switchbox we even find a better solution. For one of two examples (modified dense switchbox) for which a Manhattan solution does not exist, the wiring length of the best 2-layer solution is by a value of 2 shorter than the one of the optimal knock-knee solution. For dense switchbox, we are not aware of any feasible routing that can be realized on two layers.

Chapter 4

A Multicommodity Flow Problem

4.1 Introduction

A classical type of problems in combinatorial optimization that gives rise to integer programs with block structure are multicommodity flow problems. Given some graph $G = (V, E)$ with edge capacities $c_e, e \in E$, and possibly edge weights $w_e, e \in E$, and a set of demands $(s_1, t_1), \ldots, (s_k, t_k)$ of sizes $d^1, \ldots, d^k$, find integral flows (or paths) f_i of size d^i for $i = 1, \ldots, k$ such that the capacity constraints are met (i. e., the sum of the flows using an edge must not exceeds its capacity) and, if edge weights are given in addition, the sum of the weights of the flows is minimized.

Multicommodity flow problems are widely studied in the literature, see, for instance, Lomonosov [1985], Frank [1990], Schrijver [1990], Ahuja, Magnanti, and Orlin [1993] for interesting results and excellent surveys. Multicommodity flow problems lead to integer programs with block structure by introducing flow variables x_e^i counting the flow of demand i over edge e. The blocks then model the individual flows, see (A.1), and the linking constraints are the capacity constraints $\sum_{i=1}^k x_e^i \leq c_e$ for each edge $e \in E$.

The multicommodity flow problem is also inherited by many problems with practical applications, but usually not in its "pure" form as described here, see, for instance, Stoer and Dahl [1994], Magnanti, Mirchandani, and Vachani [1995], Bienstock and Günlük [1996], Löbel [1997], or Alevras, Grötschel, and Wessäly [1998]. The (real-world) problems might on the one hand result from generalizations of the multicommodity flow problem. We have discussed one such example, the Steiner tree packing problem, in Chapter 3. If we require all nets to have cardinality two and set the demands to one for all nets, the Steiner tree packing problem is a multicommodity flow problem. The other type of practical problems are multicommodity flow problems where further side constraints and requirements on the flows have to be satisfied.

In this chapter we deal with a real-world problem of the second type that we encountered in telecommunication. We start by describing the application and our motivation for studying this particular (multicommodity flow) problem.

A major trend in telecommunications is increased flexibility in terms of network configuration and resource allocation. In particular communication paths in networks may be set up on a temporary basis and controlled by software in order to meet changing demands due to, e.g., data communications or video applications. Such paths (often called virtual paths) have the attractive feature of low processing time in the intermediate nodes. An important problem area concerns the management of these capacitated paths, and we are concerned with such a problem in a two-layered network.

The model we study is as follows: One is given a set of point-to-point traffic demands that need to be routed in a so-called pipe-network. Each edge in this network is called an express pipe. It has a fixed, uniform capacity measured in the same units as the traffic demands. Each express pipe corresponds to a path in an underlying physical transmission network. When an express pipe is established, it uses resources in the transmission network, say, a fiber pair in a fiber cable. For each edge in the transmission network, one has therefore an upper bound on the number of pipes that can go through it. The problem is now to select some of the given express pipes such that the traffic can be routed upon them, taking into account express pipe capacity and physical link capacity. Costs are associated with the establishment of express pipes and with the routing. When we use the term "routing", we don't mean dynamic routing at call setup time. We focus rather on the setup of the express pipes which accommodate forecasted traffic and are not changed every few minutes. We also assume that the set of pipes to choose from is given beforehand. Pipes are not generated dynamically in the course of the algorithm.

One motivation for studying the routing and path-packing model comes from routing and grouping in the PDH or SDH bandwidth hierarchy. There traffic, given in units of 2 Mbit/sec, is switched onto systems of different fixed bandwidths. A model involving several levels of networks and an LP-based solution method is described in Lorentzen [1994].

Another application may be in ATM-networks. There traffic corresponds to virtual circuits, which can be packed into virtual paths (our express pipes). Our model should, however, be refined to capture this case better. Virtual paths take many bandwidths in the physical network, not just one as in our model, and our cost function does not exactly model the gains (less call control in intermediate nodes) versus the disadvantages (splitting of bandwidth) of setting up virtual paths.

Park, Kang, and Park [1994] and Parker and Ryan [1994] describe integer programming algorithms for routing (unsplittable) demands in a capacitated network so as to maximize revenue and route as many demands as possible. This is the bandwidth packing problem. Our model is distinguished from theirs in that our model involves the intermediate pipe layer, and the demand routing is modeled with flow variables instead of path variables.

This chapter is organized as follows. In Section 4.2 the integer linear programming model for the mentioned problem is presented. The body of this work is a polyhedral study in Section 4.3. Various classes of joint facet-defining inequalities are introduced. Section 4.4 discusses the separation algorithms for the described classes of inequalities and presents a primal heuristic. In the last section we report on our computational results for some realistic problems.

4.2 Mathematical Model

In this section we give a mathematical formulation of the problem, describe it as an integer linear programming model and introduce an associated polytope corresponding to the feasible solutions. Some basic properties of the polytope are discussed.

The physical network of interest is modeled as an undirected graph $N = (V, L)$ with node set V corresponding to switching nodes and edge set L corresponding to transmission lines (fiber cables). We call N the *physical graph* and its edges *physical edges (links)*. The traffic demands are modeled by the *demand graph* $D = (V, K)$

where each *demand edge* $[u^k, v^k] \in K$ represents a traffic demand between the endnodes u^k and v^k of size d^k. (Typically, there are several isolated nodes in the demand graph). The final element of our model is the *pipe graph* $G = (V, E)$ where each *pipe (edge)* $e = [u, v] \in E$ corresponds to a $[u, v]$-path in the physical graph N. A pipe may then represent a transmission path in the telecommunication network (possibly set up for a limited time period) on which different traffic may be routed. Note that G may contain many parallel edges. One may view the whole network architecture as a two-level hierarchy. Sometimes more than two levels are of interest, but we do not treat this case here.

The model also incorporates capacities in the following way. Each demand should be routed in the pipe graph, i. e., each demand $k = [u, v]$ uses some $[u, v]$-path $e_1, \ldots, e_t$ of pipe edges in G. We assume that the capacity of each pipe $e \in E$ is a constant $B > 0$ meaning that the total demand that may be routed on each pipe may not exceed B. Furthermore, the number of selected pipes (in a feasible solution) containing a physical link $l \in L$ must not exceed the capacity c_l (we assume throughout that $c_l \geq 1$). This may correspond to the situation where each pipe is allocated to an individual fiber pair on the fiber cable $l \in L$. Thus we have capacity constraints in both levels of the network architecture, both for "embedding" demands (connections) in the pipe graph, and for embedding pipes in the physical network.

The problem of interest is to select pipes that are to be used and to determine on which path of the selected pipe set each of the demands should be routed. The cost function is the sum of the costs γ_e for selecting a pipe e and the costs ω_e^k for routing a demand k through pipe e. We call this problem of finding a minimum cost pipe selection and routing the *pipe selection and routing problem* PIPE.

Note that if we neglect the pipe selection problem and are just interested in the routing part the PIPE problem is a "pure" multicommodity flow problem as discussed in the introduction with the additional restriction that the flows have to be paths.

The PIPE problem can be shown to be $\mathcal{NP}$-hard as it contains the edge-disjoint path packing problem (see Kramer and van Leeuwen [1984]) as a special case.

We model the PIPE problem mathematically as the following integer linear program

$$
\text{(4.1)} \qquad
\begin{array}{rll}
\min & \displaystyle\sum_{e \in E} \gamma_e y_e + \sum_{k \in K} \sum_{e \in E} \omega_e^k x_e^k & \\
(i) & x^k(\delta_G(W)) \geq 1, & \text{for all } W \subset V \text{ with } u^k \in W,\ v^k \notin W, \\
 & & k \in K; \\
(ii) & \displaystyle\sum_{k \in K} d^k x_e^k \leq B y_e, & \text{for all } e \in E; \\
(iii) & \displaystyle\sum_{e : l \in e} y_e \leq c_l, & \text{for all } l \in L; \\
(iv) & 0 \leq x_e^k \leq 1, 0 \leq y_e \leq 1, & \text{for all } k \in K,\ e \in E; \\
(v) & x_e^k,\ y_e \text{ integer} & \text{for all } k \in K,\ e \in E.
\end{array}
$$

The 0/1 variable y_e indicates whether pipe $e \in E$ is selected, and the variable x_e^k indicates if demand $k \in K$ uses (is routed on) pipe $e \in E$. Constraint (i) assures that $x^k \in \mathbb{R}^E$ is the incidence vector of a pipe set containing a $[u^k, v^k]$-path for each $[u^k, v^k] \in K$. Observe that these inequalities coincide with the Steiner cut inequalities (3.1) (i) if the corresponding terminal set has cardinality two. Constraints (ii) and (iii) reflect the capacity constraints in the pipe graph and

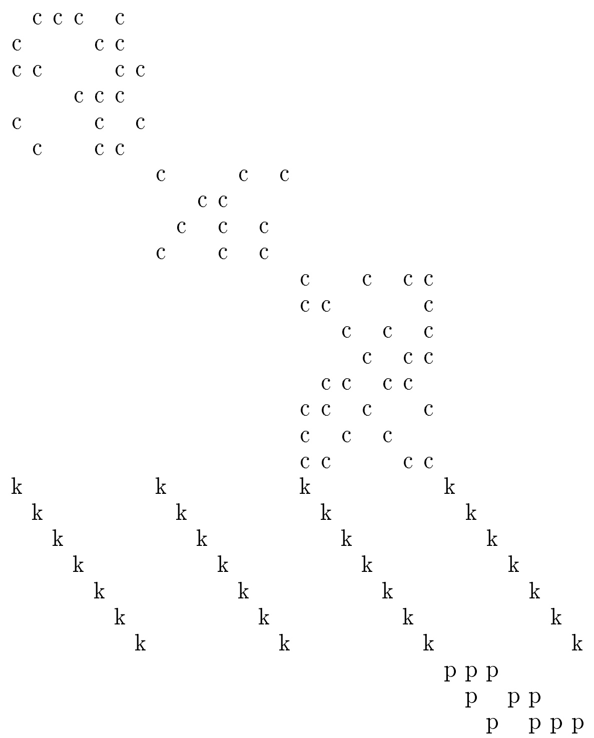


Figure 4.1: Structure of a PIPE integer program with $|E| = 7, |K| = 3$ and $|L| = 3$. The letter 'c' indicates a non-zero entry in an inequality from (4.1) (i), which are of set covering type. 'k' gives a non-zero entry of the knapsack constraints (4.1) (ii), and 'p' reflects a non-zero entry for the inequalities (4.1) (iii), which are of set packing type.

the physical graph, respectively. Figure 4.1 shows the structure of the constraints (4.1) (i) – (iii).

Let us denote by MCF $(N, G, D, B, d, c, \gamma, \omega)$ an instance of the PIPE problem, where $N = (V, L)$, $G = (V, E)$, $D = (V, K)$ are the physical, the pipe, and the demand graph, B the pipe capacity, $c = (c_l)_{l \in L}$ and $d = (d^k)_{k \in K}$ are the capacity and demand vectors, and $\gamma = (\gamma_e)_{e \in E}$ and $\omega = (\omega_e^k)_{e \in E, k \in K}$ are the pipe and routing cost vectors. If it is clear from the context we abbreviate an instance by MCF. We introduce a family of integer polytopes associated with the model in (4.1):

$$P_{\mathrm{MCF}}\,(N, G, D, B, d, c) := \operatorname{conv}\{\, (x, y) \in \mathbb{R}^{K \times E} \times \mathbb{R}^{E} : (x, y) \text{ satisfies (4.1) (i) – (v)} \,\}, \tag{4.2}$$

where we write (x, y) instead of $(x^T, y^T)^T$ for convenience. Again, if it is clear from the context we abbreviate $P_{\mathrm{MCF}}\,(N, G, D, B, d, c)$ by P_{MCF}. The polytope P_{MCF} has a complicated polyhedral structure, and an analysis of some of its properties is given in the next section.

We shall assume from now on that the demand set K is partitioned into two subsets K_1 and K_2 such that $d^k = 1$ for $k \in K_1$ and $d^k = B$ for $k \in K_2$. (When $B = 1$, we let $K_1 = \emptyset$ and $K_2 = K$.) This is of interest in SDH-applications, where a fixed number of systems of low bandwidth fits into a system of higher bandwidth. Furthermore, in our implementation and numerical experiments we have restricted our attention to the parameter choice $\gamma_e = \gamma$ for all $e \in E$ and $\omega_e^k = d^k \omega_e$. If ω_e is the number of physical links in pipe e, then γ can be interpreted as follows. If we have the choice between installing a new direct pipe for a demand of value one and using the spare capacity of an existing path of length less than γ in the physical network, then the "long" path is preferred.

4.3 Polyhedral Properties

The goal of this section is to establish a number of properties of the polytope P_{MCF}. We study the dimension of P_{MCF} and additional classes of (joint) inequalities that define facets of this polytope. Let an instance MCF $(N, G, D, B, d, c, \gamma, \omega)$ be given.

The problem of deciding whether P_{MCF} is non-empty (i.e., finding a feasible solution in (4.1)) is $\mathcal{NP}$-complete. This follows from the fact that the special case of deciding the existence of edge-disjoint paths between specified terminals is $\mathcal{NP}$-complete, see Kramer and van Leeuwen [1984]. However, as in the case of the Steiner tree packing problem (cf. Lemma 3.2.1) we can work out conditions under which P_{MCF} is full dimensional.

Proposition 4.3.1 *P_{MCF} is full dimensional if the PIPE instance $MCF(e) = MCF(N, (V, E \setminus \{e\}), D, B, d, c', \gamma, \omega)$ is feasible for each $e \in E$, where $c'_l = c_l$ for all $l \notin e$ and $c'_l = c_l - 1$ for all $l \in e$.*

Proof. Assume that MCF(e) has a feasible solution for each $e \in E$. Also assume that P_{MCF} is contained in the hyperplane defined by the linear equation

$$\sum_{e \in E} a_e y_e + \sum_{k \in K} \sum_{e \in E} b_e^k x_e^k = \alpha. \tag{4.3}$$

Let $e \in E$. By assumption there is a feasible solution (x, y) in (4.1) with $y_e = 0$ and with capacity function c'. Define $y' \in \mathbb{R}^E$ by $y'_f = y_f$ for $f \neq e$ and $y'_e = 1$. For each $l \in e$ we then have $\sum_{e' : l \in e'} y'_{e'} \leq c'_l + 1 = c_l$, and it follows that (x, y') is feasible. Thus both (x, y) and (x, y') satisfy (4.3) and this implies that $a_e = 0$. As e was arbitrary, we get $a = 0$. Furthermore, let x' be obtained from x by setting $(x')_e^k = 1$ for some k. Then both (x, y') and (x', y') are feasible in (4.1) and therefore satisfy (4.3). This leads to $b_e^k = 0$ for all $k \in K$ and $e \in E$. Thus $a = 0$ and $b = 0$, which contradicts that the inequality in (4.3) defines a hyperplane. Therefore $P_{\mathcal{S}}$ is full dimensional as claimed. □

We remark that in the case when $L = E$, a necessary condition for P_{MCF} to be full dimensional is that MCF(e) is feasible for all $e \in E$.

All the inequalities in (4.1) define facets of P_{MCF} whenever the pipe graph is "dense" enough. We do not go into these details. Note that the only non-trivial individual inequalities are the ones in (4.1) (i), because these inequalities completely describe the dominant of the path polytope conv$\{\chi^P : P$ is a path from u^k to $v^k\} + \mathbb{R}_+^E$, for each demand $k \in K$. This situation is different to the ones discussed in Chapter 2 and 3, where complete descriptions for the polytopes associated with individual blocks are not at hand. That means that all new classes of (facet-defining) inequalities we are discussing in the following are joint inequalities.

4.3.1 Knapsack Inequalities

Each inequality in (4.1) (ii) may be viewed as a knapsack inequality. In fact, complementing variables $y_e, e \in E$, i.e., using the linear transformation $T_e(y) = z$ where $z_e = 1 - y_e$, for each $e \in E$, we get the knapsack inequality

$$\sum_{k \in K_1} x_e^k + B \sum_{k \in K_2} x_e^k + B z_e \leq B. \tag{4.4}$$

Each valid inequality for the knapsack polytope defined by (4.4) is also valid for P_{MCF} when setting $z_e = 1 - y_e$. One particular class of inequalities that is valid for the knapsack polytope defined by (4.4) are the cover inequalities, see the summary in Section 2.3. Lifted cover inequalities read in our case as follows:

$$x_e^{k_1} + \sum_{k \in K_2} x_e^k \leq y_e \quad \text{for all } k_1 \in K_1. \tag{4.5}$$

A combinatorial interpretation of such an inequality is that if more than one demand is routed on e, then all these demands are K_1-demands. In certain special situations a complete linear description of knapsack polytopes has been found, see Weismantel [1997]. It follows from the results of Weismantel [1997], see also page 12, that a complete linear description of the knapsack polytope defined by (4.4) is given by the inequalities (4.4), (4.5) and the trivial inequalities.

Note that if $|K_1| \leq B$, then the knapsack inequality (4.1) (ii) is dominated by the sum of cover inequalities. Under certain known conditions the cover inequalities define facets of the knapsack polytope (see again Section 2.3). With suitable additional assumptions on the PIPE instance MCF, the cover inequalities also define facets of P_{MCF}.

4.3.2 Strengthened Cut Inequalities

Consider a cut $\delta_G(W)$ in the pipe graph, where W and $V \setminus W$ are non-empty. Let K' be the demands in $K_1 \cap \delta_D(W)$. Then

$$y(\delta_G(W)) - \sum_{k \in K_2} x^k(\delta_G(W)) \geq \lceil |K'|/B \rceil \tag{4.6}$$

is a valid inequality. To see this add the valid inequalities

- $x^k(\delta_G(W)) \geq 1$ for all $k \in K'$
- $By_e - \sum_{k \in K_1} x_e^k - B \sum_{k \in K_2} x_e^k \geq 0$ for all $e \in \delta_G(W)$

and divide the result by B. Then round the coefficients of the left-hand side of this new inequality by adding an appropriate amount of $x_e^k \geq 0$, and round the right-hand side. The resulting valid inequality is (4.6) which we call the *strengthened cut inequality*. These inequalities can also be shown to be non-redundant for highly-connected graphs G and N.

Theorem 4.3.2 *The strengthened cut inequality (4.6) defines a facet of P_{MCF} if the following conditions are satisfied.*

(i) $|K'|/B$ is not an integer and $|K'| > B$.

(ii) There are at least $\max\{\lceil d(\delta_D(w))/B \rceil + 1 : w \in \{u, v\}\}$ parallel $[u, v]$-pipes between any $u, v \in V$, $u \neq v$.

(iii) Each pipe has length one in N, and using all pipes does not violate the capacity constraints in N.

Proof. First, P_{MCF} is full dimensional, because the conditions of Proposition 4.3.1 are satisfied. Consider a facet-defining inequality

$$\sum_{e \in E} a_e y_e + \sum_{k \in K} \sum_{e \in E} a_e^k x_e^k \geq \alpha \tag{4.7}$$

such that each root of (4.6) satisfies (4.7) with equality. As P_{MCF} is full dimensional, it suffices to show that the two inequalities (4.6) and (4.7) are equal up to a positive scalar multiple.

For any set $F \subseteq \delta_G(W)$ of cardinality $t := \lceil (1/B)\, d(\delta_D(W)) \rceil$ there exists a root with $y_e = 1$ for $e \in F$ and $y_e = 0$ for $e \in \delta_G(W) \setminus F$, where all demands in $D[W]$ and $D[V \setminus W]$ are routed on a direct pipe, and where all remaining demands $[u, v]$ with $u \in W$ and $v \notin W$ are routed one by one on at most three pipes $[u, r]$, $[r, s]$ and $[s, v]$. Here $r \in W$ and $s \notin W$ are chosen such that $[r, s] \in F$, and $[r, s]$ still has free capacity. $r = u$ and $s = v$ is allowed. Demands in K_2 should be routed before demands in K_1. Pipes $[u, r]$ and $[s, v]$ exist due to assumption (ii). Capacity constraints in the physical network are not violated due to assumption (iii). The described solution satisfies (4.6) and hence (4.7) with equality.

If $E(W)$ is non-empty, then, by (ii), it contains at least one pipe not used by the described root solution, so one can prove that $a_e = 0$ by setting $y_e = 1$. In a similar manner one can prove $a_e^k = 0$ for all $e \in E(W) \cup E(V \setminus W)$ and all $k \in K$. Because of condition (i) one pipe e in F still has a free capacity of at least one in the root solution using F, and, moreover, there is one more pipe f which contains demands of size one. This allows to prove $a_e^k = 0$ for any of the demands k currently routed through f. By exchanging e and f and rerouting the small demands one can prove $a_e^k = 0$ for $e \in \delta_G(W)$ and $k \in K_1$. Now compare a root solution using $F \subseteq \delta_G(W)$ with a root solution using $F \setminus \{e\} \cup \{f\}$ for arbitrary edges $e \in F$ and $f \in \delta_G(W) \setminus F$. The routings in $F \setminus \{e\}$ are supposed to be the same in both solutions, and e is supposed to carry only demands of $K' = K_1 \cap \delta_D(W)$. Note that, because of condition (i), K' is not empty. Comparison of the two solutions proves that $a_e = a_f$. This is true for all $e, f \in \delta_G(W)$. Now compare a root solution using $F \subseteq \delta_G(W)$ with the root solution in which an arbitrary edge $e \in \delta_G(W) \setminus F$ is added to F, then y_e and some x_e^k for $k \in K_2$ are set to one, and all other variables stay the same. This proves that $a_e^k = -a_e$. Since e, k, and F were arbitrary, (4.7) has the same coefficients as the strengthened cut inequality (4.6), hence it defines a facet. □

Note that condition (i) is also necessary for (4.6) to define a facet. Conditions (ii) and (iii) are only of technical interest. They can probably be replaced by milder conditions on the connectedness of G and N.

Remark 4.3.3 *The strengthened cut inequalities may be generalized in the spirit of the "flow-cutset inequalities" introduced in Bienstock, Chopra, Günlük, and Tsai [1998]. Let F be a subset of $\delta_G(W)$. In the validity proof above the inequalities $x^k(\delta_G(W))$ are added as before, but now the knapsack inequalities are added only for $e \in F$. The resulting* flow-cutset inequality *is*

$$y(F) + \sum_{k \in K'} x^k(\delta_G(W) \setminus F) - \sum_{k \in K_2} x^k(F) \geq \lceil |K'|/B \rceil.$$

4.3.3 Hypomatchable Inequalities

We introduce and study a large class of inequalities called hypomatchable inequalities.

Consider an instance MCF $(N, G, D, B, d, c, \gamma, \omega)$ of PIPE with $B \geq 2$. Choose an odd number of nodes $V' = \{v_1, v_2, \ldots, v_n\} \subseteq V$, and demands $k_1, k_2, \ldots, k_n$ in

K_1 (not necessarily distinct) such that demand k_i is incident to v_i. Lastly, choose a set $F \subseteq E(V')$ with the property that if $k_i = k_j$ then $[v_i, v_j]$ is not in F. Denote by K' the set of chosen demands with only one endpoint in V', and denote by K'' the set of chosen demands with two endpoints in V'. Let F' be the set F together with all edges $[v_i, v_j]$ with $k_i = k_j$. Consider the inequality

$$y(F) - \sum_{k \in K_2} x^k(F) + \sum_{i=1}^{n} x^{k_i}(\delta_G(v_i) \setminus F') + \sum_{k \in K''} \sum_{\substack{e \in E: \\ e \text{ connects} \\ u^k \text{ and } v^k}} x_e^k \geq \lceil n/2 \rceil, \tag{4.8}$$

which we call a *hypomatchable inequality*, because, as we shall see later, the inequality has a good chance to be facet-defining when (V', F') defines a hypomatchable graph.

Lemma 4.3.4 *The hypomatchable inequality (4.8) is valid for P_{MCF}.*

Proof. Add the valid degree and cover inequalities

- $x^{k_i}(\delta_G(v_i)) \geq 1$ for $i = 1, \ldots, n$ and
- $y_e - x_e^{k_i} - \sum_{k \in K_2} x_e^k \geq 0$ and
  $y_e - x_e^{k_j} - \sum_{k \in K_2} x_e^k \geq 0$ for each $e = [v_i, v_j] \in F$.

Divide both sides by two, and round up all coefficients on the left-hand side by adding the corresponding non-negativity constraints $\frac{1}{2}x_e^k \geq 0$. Since the left-hand side takes integer values for all $(x, y) \in P_{\text{MCF}}$, one can round up the right-hand side to get a valid inequality, namely (4.8). □

As an illustration, consider a three-node example with nodes v_1, v_2, v_3 and parallel pipes e_i and e_i' both with endnodes v_i and v_{i+1} for $i = 1, 2, 3$ ($v_4 = v_1$). We also let $L = E$, $B = 4$. Demand k is parallel to e_k and $d^k = 1$ for $k = 1, 2, 3$. Let $F := \{e_1, e_2, e_3\}$ and define the fractional solution $(\bar{x}, \bar{y})$ by $\bar{x}_e^k = \bar{y}_e = 1/2$ if $e \in F$ and $\bar{x}_e^k = \bar{y}_e = 0$ for $e \in E \setminus F$. This solution corresponds to the non-integral routing of each demand by splitting the flow equally along the two paths between each pair of nodes on the triangle. One can verify that $(\bar{x}, \bar{y})$ satisfies all the linear inequalities in (4.1) as well as the knapsack and cover inequalities (4.4), (4.5), and the strengthened cut inequalities (4.6). However, $(\bar{x}, \bar{y})$ violates the hypomatchable inequality $y(F) + \sum_{i=1}^{3} x^{k_i}(\delta_G(v_i) \setminus F) \geq 2$.

We discuss conditions under which a hypomatchable inequality is facet-defining. We introduce some convenient terminology. For a graph $H = (V, F)$ with an odd number of nodes, we call $M \subseteq F$ a *supermatching* if all nodes except one are incident to exactly one edge of M, and the last node is incident to two edges of M. A supermatching of H has $(|V| + 1)/2$ edges.

Consider a root (x, y) of a hypomatchable inequality (4.8), i.e., (x, y) is a feasible solution of (4.1) that satisfies (4.8) with equality. Let $M := \{e = [v_i, v_j] \in F' : x_e^{k_i} = 1 \text{ or } x_e^{k_j} = 1\}$. It can be seen that there is at most one isolated node in (V', M), and that M is either a supermatching with $\lceil n/2 \rceil$ or a matching with $\lfloor n/2 \rfloor$ edges.

A graph $H = (W, F)$ is *hypomatchable* (see Lovász and Plummer [1986]) if $H[W \setminus \{v\}]$ contains a perfect matching for each $v \in W$. Examples of hypomatchable graphs include odd cycles and the complete graph on an odd number of nodes.

Remark 4.3.5 *Every hypomatchable graph is connected.*

Theorem 4.3.6 *A hypomatchable inequality (4.8) defines a facet of P_{MCF} if the following conditions hold:*

(i) $L = E$, $K = K' \cup K''$;
(ii) G *is a complete graph and* $|V'| < |V| - 1$;
(iii) $G_{F'} = (V', F')$ *is hypomatchable.*

Proof. Since the conditions of Proposition 4.3.1 are satisfied, P_{MCF} is full dimensional.

Consider a facet-defining inequality

$$\sum_{e \in E} a_e y_e + \sum_{k \in K} \sum_{e \in E} a_e^k x_e^k \geq \alpha \tag{4.9}$$

such that each root of (4.8) satisfies (4.9) with equality. As P_{MCF} is full dimensional, it suffices to show that the two inequalities (4.8) and (4.9) are equal up to a positive scalar multiple.

We first describe a basic construction of roots of (4.8). Let M be a matching of $G_{F'}$ of size $\lfloor n/2 \rfloor$ or a supermatching of $G_{F'}$ of size $\lceil n/2 \rceil$. Set $y_e = 1$ for $e \in M$ and $y_e = 0$ for $e \in F' \setminus M$. If v_i is incident to at least one edge in M, route k_i such that $x^{k_i}(\delta_G(v_i) \setminus M) = 0$. If v_i is not incident to an edge in M (there can be at most one such node) route k_i such that $x^{k_i}(\delta_G(v_i) \cap F') = 0$. With $L = E$ and G complete it is always possible to find such a routing. Then (x, y) is feasible and a root of (4.8).

For each $e \in E \setminus F'$ one can construct a root (x, y) of (4.8) with $y_e = 0$ by choosing a supermatching in the basic root construction and avoiding e in the routing. This works because of $|V'| < |V| - 1$. For each $e \in F' \setminus F$ one can construct a root (x, y) of (4.8) with $y_e = 0$ by choosing a maximum matching M of $G_{F'}$ that avoids e. That is possible by condition (iii). By comparing these solutions with the corresponding root solutions where y_e is set to one, one proves that $a_e = 0$ for each $e \in E \setminus F$. Similarly, we derive $a_e^k = 0$ for $e \in E \setminus F$ and for those $k \in K$ whose coefficient in (4.8) is zero.

For given $e = [v_i, v_j] \in F$ choose a perfect matching of $G_{F'}[V' \setminus \{v_i\}]$ and augment it to a supermatching M by adding edge e. With the basic construction one may now create a root with $y_e = 1$, $x_e^{k_i} = 1$ and no other demand using e. By setting $x_e^k = 1$ for some $k \neq k_i$ one obtains a new root. (Note that K contains only small demands, because of the condition $K = K' \cup K''$.) This proves $a_e^k = 0$. Since e and k are arbitrary, and, moreover, $k_i \neq k_j$, one gets $a_e^k = 0$ for all $e \in F$ and $k \in K$.

For $e = [v_i, v_j] \in F' \setminus F$ one can similarly prove that $a_e^k = 0$ for all $k \neq k_i$ $(= k_j)$.

Thus, whenever in (4.8) a coefficient of some variable is zero, then the corresponding coefficient in (4.9) is zero.

Let $e = [v_i, v_j] \in F$ and $f \in \delta_G(v_i) \setminus F$. We shall prove that $a_e = a_f^{k_i}$. Pick a perfect matching M in $G_{F'}[V' \setminus \{v_i\}]$. If $f \in F'$, augment this matching by edge f. The basic routing construction can be done such that demand k_i is routed on edge f. Compare this solution to the one where demand k_i uses edge e instead of f. We get $a_e = a_f^{k_i}$ for any e, f and k_i chosen as above. If $e \in F' \setminus F$ and $f \in \delta_G(v_i) \setminus F'$, a similar construction shows $a_e^{k_i} = a_f^{k_i}$.

Now let e, f, and g be three edges with endnode v_i. When e and f are in F and $g \notin F$ we have shown $a_e = a_g^{k_i} = a_f$. When $e \in F$, $f \in F' \setminus F$, and $g \notin F'$ we have $a_e = a_g^{k_i} = a_f^{k_i}$. Note that $e, f \in F' \setminus F$ is not possible. By Remark 4.3.5 $G_{F'}$ is connected, and thus (4.9) is a scalar multiple of (4.8), showing that (4.8) defines a facet of P_{MCF}. □

We note that conditions (i) and (ii) are present only to simplify the proof. Either of them can be relaxed. Especially the restriction on the number and size of demands is not necessary. Condition (iii) is probably necessary, but we have not been able to prove this.

In our computations we have chosen F to be an odd cycle. We call this subclass of (4.8) *cycle inequalities.*

The hypomatchable inequalities may be extended into larger classes of facet-defining inequalities using lifting techniques, see Appendix C. The idea is to shrink certain node sets in some PIPE instance and thereby obtain a "smaller" related instance for which a hypomatchable inequality is valid. The lifted inequality is obtained by letting all edges that were shrunk get a coefficient zero. One can show (under certain conditions on the subgraphs that are shrunk) that a lifted hypomatchable inequality is non-redundant.

4.4 Algorithmic Aspects

In this section we discuss our separation algorithms and primal heuristic that are incorporated in the branch-and-cut algorithm. In all other aspects our implementation basically follows the general description of a branch-and-cut algorithm given in Appendix B. In the following let V' be the subset of V consisting of all endnodes of demand edges (i. e., nodes u^k and v^k for $[u^k, v^k] \in K$).

4.4.1 Separation Algorithms

In the following we discuss separation algorithms for the cut inequalities (4.1) (i) and (4.6), the cover inequalities (4.5), and the cycle inequalities (4.8).

As initial cuts, i. e., those cuts that set up the first LP, we use the trivial inequalities $0 \leq x_e^k \leq 1$, $0 \leq y_e \leq 1$, and the degree constraints $x^k(\delta_G(u^k)) \geq 1$ and $x^k(\delta_G(v^k)) \geq 1$ for $[u^k, v^k] \in K$. In addition, we add some of the strengthened cut inequalities in the following way. For each node $v \in V'$, we check whether $(d(\delta_D(v))$ modulo $B) \neq 0$. If that is the case, we add the corresponding strengthened cut inequality to the initial LP. If not, we try to extend the node set $W = \{v\}$ in a greedy like fashion (by checking all neighbouring nodes of W) until we find a set W satisfying $(d(\delta_D(W))$ modulo $B) \neq 0$ (in this case we add the strengthened cut inequality induced by W to the initial LP) or the list of neighbours is empty. This set of inequalities represents the first LP.

Let $(\bar{x}, \bar{y})$ denote the actual LP solution. Since we add to the initial LP all trivial inequalities, we can suppose in the following that all components of $(\bar{x}, \bar{y})$ are non-negative and less than or equal to one.

Cut inequalities

In order to solve the separation problem for the cut inequalities (4.1) (i) for a particular demand $k \in K$, we have to decide whether the minimum cut capacity between u^k and v^k is less than one where edge capacities are given by $\bar{x}^k$. This can be done using any max-flow algorithm. We implemented the highest-label preflow push algorithm suggested by Goldberg and Tarjan [1988] (see also Ahuja, Magnanti, and Orlin [1993]). This algorithm runs in time $O(|V'|^2\sqrt{|E|})$.

We also use our max-flow algorithm to find violated strengthened cut inequalities (4.6). We assign each edge $e \in E$ the capacity $\max\{0, \bar{y}_e - \sum_{k \in K_2} \bar{x}_e^k\}$, and determine for each $[u^k, v^k] \in K_1$ a minimum $[u^k, v^k]$-cut, $\delta_G(W^*)$ say, for $W^* \subset V$. If $|\delta_D(W^*) \cap K_1|/B$ is not integer, we add the corresponding strengthened cut inequality in case it is violated.

Cover inequalities

Separating the cover inequalities (4.5) is easy. Since the number of cover inequalities is linear (in the number of demands and edges, see Section 4.3), we sequentially check all of them for possible violation. Note that the total number of cover inequalities may be very large (see next section), but only a fraction of them is needed to solve the problem.

Cycle inequalities

We do not know whether the separation problem for the cycle inequalities (4.8) can be solved in polynomial time (when F defines a cycle). The problem is that we do not know how to find a low-weight cycle that only contains terminal nodes for demands of value one. In the following we present a heuristic, where we first try to find a minimum cycle with respect to a certain weight function, and then choose, if possible, for each node in the cycle, the best demand of size one.

We first describe the weight function according to which we want to find a minimum cycle. Consider again (4.8) with F being a cycle with node set $\{v_1, \dots, v_n\}$, n odd, and $k_1, \dots, k_n$ demands of size one such that demand k_i terminates at node v_i $(i = 1, \dots, n)$. Since the objective function is non-negative, the optimum integer solution and also the current LP solution satisfy $x^{k_i}(\delta(v_i)) = 1$. If we subtract this equation (for $i = 1, \dots, n$) from the cycle inequality, we get

$$y(F) - \sum_{k \in K_2} x^k(F) - \sum_{i=1}^{n} x^{k_i}(\delta_G(v_i) \cap F') + \sum_{k \in K''} \sum_{\substack{e \in E: \\ e \text{ connects} \\ u^k \text{ and } v^k}} x_e^k \geq -\lfloor \tfrac{n}{2} \rfloor. \tag{4.10}$$

Suppose for a moment that we do not have parallel edges, and all demands $k_1, \dots, k_n$ are distinct. If we choose as edge weights, for $uv \in E$,

$$w_{uv} := 0.5 + \bar{y}_{uv} - \sum_{k \in K_2} \bar{x}_{uv}^k - \sum_{\substack{k \in K_1: \\ \{u^k, v^k\} \cap \{u,v\} \neq \emptyset}} \bar{x}_{uv}^k,$$

then $w(F) - 0.5$ exactly coincides with the difference between the left- and right-hand side of (4.10), when F is an odd cycle and each node in the cycle has exactly one incident demand of size one. Thus, if $w(F) < 0.5$ and F is odd, we have a violated cycle inequality, otherwise not. If there are parallel edges $\{e_1, \dots, e_p\}$, $p \geq 2$, connecting nodes u and v, we aggregate these edges to a single edge, uv say, and assign it the weight

$$w_{uv} := 0.5 + \sum_{i \in I} \Big(\bar{y}_{e_i} - \sum_{k \in K_2} \bar{x}_{e_i}^k - \sum_{\substack{k \in K_1: \\ \{u^k, v^k\} \cap \{u,v\} \neq \emptyset}} \bar{x}_{e_i}^k \Big),$$

where $I := \{i \in \{1, \dots, p\} : \text{there exists some } k \in K_1 \text{ with } \bar{x}_{e_i}^k > 0\}$.

Now we determine an odd cycle F^* with "low" weight $w(F^*)$ (see below). If some node on the cycle is incident only to demands of size B, the cycle is rejected. Note that according to the definition of w, $w(F^*) - 0.5$ is a lower bound for the slack of the most violated cycle inequality. Thus, if $w(F^*) \geq 0.5$, there is no violated cycle inequality. Otherwise, we determine for each node v_i the best possible demand k_i, i. e., we choose

$$k_i := \arg \min_{k \in K_1 : v_i \in \{u^k, v^k\}} \bar{x}^k(\delta_G(v_i) \setminus F^*).$$

It might be that the cycle inequality defined by our choices F^* and $k_1, \dots, k_{|F^*|}$ no longer yields a violated inequality, because the edge weights w_{uv} do not reflect the exact slack, when nodes have more than one incident demand of size one.

There remains the problem of finding a minimum cycle F^* in an undirected graph $G = (V, E)$ with edge weights w_{uv}, $uv \in E$, that may be negative. If the edge weights are indeed arbitrary, the problem to determine a minimum cycle is $\mathcal{NP}$-complete. However, we can decide whether there is a cycle of negative weight and if not, find a minimum cycle by transforming the problem to a perfect matching problem (see Ahuja, Magnanti, and Orlin [1993]). Thus, we can decide whether there exists a cycle of weight less than 0.5 which might give rise to a violated inequality. Since a perfect matching algorithm is very time consuming and since such an algorithm might return just one cycle, we preferred to implement the following heuristic. Starting from each node $v \in V$, we determine a shortest spanning tree by using Prim's algorithm (Prim [1957]) and check all fundamental cycles to determine whether their weight is less than 0.5. If the cycle is even, we contract one edge. This results in many violated cycle inequalities, and for many instances this algorithm finds a cycle of weight less than 0.5 whenever there is one.

4.4.2 The Primal Heuristic

Our primal heuristic is an iterative rounding heuristic, i. e., the idea is to fix a set of fractional variables of the current LP solution to zero or one, solve the LP again, and iterate this process until all variables are integer. The tuning parameters of this heuristic are the order in which the fractional variables should be fixed in one step and their number. It turned out that the heuristic in general worked best when we just fix one fractional variable at a time and choose a fractional variable that is close to one. Moreover, we fix all variables that are one to value one for the rest of the heuristic.

If the heuristic does not change the linear program (except for fixing variables) it frequently ends with an integral solution that violates one of the cut or cover constraints in (4.1), since not all of these constraints are contained in the LP. Therefore we separate those inequalities for each fractional solution appearing in the course of the heuristic. This unfortunately slows the heuristic down. In order to speed up the separation process in the heuristic, we only add those cut inequalities (4.1) (i), knapsack inequalities (4.1) (ii) and cover inequalities (4.5) that are violated by at least 0.5 (the usual violation epsilon in the cutting plane phase is 0.1). Moreover, we restrict the number of times the heuristic is called, depending on its success. More precisely, we calculate (a) the ratio between the time spent in the heuristic and the total time, and (b) the ratio between the number of times the heuristic could improve the best solution and the number of times the heuristic was called. If the "time" ratio is less than the "success" ratio, we call the heuristic, otherwise not. Note also that we do not call the primal heuristic after each LP, but only after the cutting plane phase for the current node is finished, i. e., we have not found any more violated inequalities, and the current LP solution is fractional. The results in the next section show that this strategy performs quite well. We obtain reasonably good primal solutions by spending at most 30% (usually less than 10%) of the total time in the heuristic.

4.5 Computational Results

In this section we report on the test runs performed with our branch-and-cut algorithm. The code is implemented in C, and all results were obtained on a Sun SPARC 20 Model 71. The examples are modified real-world examples with pipe capacity $B = 4$. Table 4.1 summarizes the data. Column 2 and 3 show the number of nodes and links of the physical network, Columns 4 and 5 contain the corresponding information for the pipe graph. Here, $|V'|$ is the number of nodes incident to some demand edge. Columns 6 and 7 give the number of demands of size one and four. The last column gives the number of 0/1 variables in our IP formulation. The numbers range from about 250 for the smallest problem up to 25000 variables. The capacities c_l in the physical network (not in the table) vary from 2 to 12 for the "nw"-examples and from 4 to 48 for the "terb"-examples.

The test series in Table 4.1 are based on two physical networks. nw is an example that approximates parts of the physical network of Norway. All other examples whose name starts with "nw" are derived from nw. nw3 differs from nw in that it contains some further physical links, that some more physical nodes are endnodes of demands, and that the set of possible express pipes is extended. If the name contains the letters ".0", the link capacities (of the example without ".0") are multiplied by ten in order to see how the link capacities influence the solution. Examples ending with ".p" have more express pipes than the corresponding example without ".p". The input pipes in the ".p"-examples were generated by finding for each demand k a set of short $[u^k, v^k]$-paths (these were determined by adding certain edges to shortest path trees). Example nw3.d1.p results from nw3.p by changing the size of nine demands from B to one. The last three examples in Table 4.1 are typical for local area networks. The demand graph of terbstar consists of node-disjoint stars. The demands of terbco form a complete graph between the root nodes of these stars, and the demand graph of terbstco is the union of these two demand graphs.

Example	N		G		Demands		Variables
	$\|V\|$	$\|L\|$	$\|V'\|$	$\|E\|$	size 1	size B	
nw	27	44	5	22	2	8	242
nw.p	27	44	5	63	2	8	693
nw3	27	60	10	91	3	18	2002
nw3.p	27	60	10	191	3	18	4202
nw3.0	27	60	10	91	3	18	2002
nw3.0.p	27	60	10	191	3	18	4202
nw3.d1.p	27	60	10	191	12	9	4202
terbco	62	81	7	113	0	21	2486
terbstar	62	81	56	248	36	12	12152
terbstco	62	81	56	359	36	33	25130

Table 4.1: Pipe selection and routing problems: Data

Unfortunately, the network planners could not give us any reasonable numbers for the cost of installing the express pipes. Thus, we performed different tests varying the installation cost γ from 0 (which means that we get the express pipes for free) up to 10 which results in rather high express pipe costs compared to the routing costs. Costs ω_e^k were set to $d^k\omega_e$, where ω_e is the number of links in pipe e.

Table 4.2 through 4.5 summarize our tests. Column 2 gives the number of inequalities of the initial LP, Columns 3 to 5 show the number of violated knapsack (cf. (4.1) (ii) and (4.5)), cut (cf. (4.1) (i) and (4.6)), and cycle (cf. (4.10)) inequalities. The number of LPs solved (including those in the primal heuristic) and the number of solved branch-and-bound nodes are presented in Columns 6 and 7. Columns 8 and 9 show the global lower bound and the value of the best feasible solution after the algorithm stopped. The total time (in CPU seconds) of the algorithm and the time spent in the heuristic are given in the last two columns.

Looking at Table 4.2 with the results for $\gamma = 0$ we see that we can solve all problem instances in the root node, i. e., we do not have to branch. Even more, with the exception of terbstar and terbstco the solutions of the root LPs are integer, since the primal heuristic has not been called. This indicates that the inequalities

Example	Cutting Planes				LPs	B&B	LB	UB	Times (sec)	
	init	knap	cuts	cycle					Heur	Total
nw	25	28	6	0	5	1	183	183	0.0	0.1
nw.p	25	71	0	0	11	1	156	156	0.0	0.2
nw3	52	125	30	18	8	1	287	287	0.0	0.6
nw3.p	51	224	3	17	19	1	243	243	0.0	1.7
nw3.0	52	63	33	4	12	1	287	287	0.0	0.5
nw3.0.p	51	46	1	4	4	1	227	227	0.0	0.4
nw3.d1.p	51	469	7	108	13	1	139	139	0.0	5.0
terbco	42	31	31	0	8	1	536	536	0.0	0.5
terbstar	150	1200	389	334	42	1	193	193	1.0	74.8
terbstco	192	3683	1194	1142	138	1	732	732	17.8	1483.0

Table 4.2: Results when installation of express pipes is for free

we separate are indeed important to solve the problems. Note that all ".p"-examples have lower objective function value than their corresponding counter part without ".p". An interesting question is how the number and variability of the express pipes influences the solutions. To completely answer this question and to find the best feasible solution among all possible express pipes, our algorithm must be embedded into a column generation approach. In case the network planners do not impose any restrictions on the set of express pipes, it will be an interesting task for the future to integrate the cutting plane and the column generation approach in order to obtain the globally best solution. We cannot draw any conclusions from this test set on whether the link capacities have an influence on the quality of the solution. For nw3 the optimum is the same, for nw3.p we obtain a better solution. A noteworthy fact is that all "nw"-examples are solved within seconds. The "terb"-examples seem to be harder, but still our algorithm provides the optimum solution after at most 25 minutes of CPU time.

For $\gamma = 1$ (see Table 4.3) the results are basically the same with slightly higher running times. But, if we further increase γ to five (Table 4.4) or to ten (Table 4.5) the situation changes. We still can solve all "nw"-examples within one minute, but for terbstar and terbstco our algorithm gets stuck. We can give a solution guarantee of 9% or less after about 3 hours of CPU time (which might be acceptable in practice), but we almost cannot improve this gap any further, even if we spend some more hours of CPU time. Since the express pipes are very expensive, the algorithm tries to avoid using y-variables. What is missing are further inequalities (like the hypomatchable inequalities) that force the y-variables to one whenever the routing variables x are positive. Further research in this area will be necessary to solve problems with larger values of γ.

Example		Cutting Planes			LPs	B&B	LB	UB	Times (sec)	
	init	knap	cuts	cycle					Heur	Total
nw	25	24	7	0	6	1	197	197	0.0	0.1
nw.p	25	57	3	0	9	1	166	166	0.0	0.2
nw3	52	120	34	34	8	1	315	315	0.0	0.7
nw3.p	51	226	8	27	14	1	264	264	0.0	1.8
nw3.0	52	113	33	33	7	1	315	315	0.0	0.8
nw3.0.p	51	146	5	21	8	1	248	248	0.0	0.9
nw3.d1.p	51	532	14	112	29	1	156	156	0.1	8.6
terbco	42	47	32	0	10	1	557	557	0.0	0.6
terbstar	150	1385	384	511	23	1	241	241	0.6	107.0
terbstco	192	3431	1087	1056	93	1	801	801	2.5	1298.7

Table 4.3: Results when installation costs for express pipes are low

Example		Cutting Planes			LPs	B&B	LB	UB	Times (sec)	
	ini	knap	cuts	cycle					Heur	Total
nw	25	24	7	0	6	1	253	253	0.0	0.1
nw.p	25	60	3	0	10	1	206	206	0.0	0.2
nw3	52	130	31	28	9	1	427	427	0.0	0.9
nw3.p	51	295	9	50	19	1	348	348	0.0	4.2
nw3.0	52	137	35	38	9	1	427	427	0.0	1.9
nw3.0.p	51	235	6	46	20	1	332	332	0.0	2.5
nw3.d1.p	51	757	15	159	31	1	217	217	0.0	16.9
terbco	42	56	26	0	8	1	641	641	0.0	0.5
terbstar	150	7821	3417	3202	3064	697	423	433	965.0	10003.3
		12821	5194	5138	4950	1303	423	433	1386.9	20008.6
		26186	10967	11332	9679	2632	424	433	1817.9	40014.0
terbstco	192	5365	1596	1833	1087	196	1064	1077	1320.2	10141.5
		7270	2295	2643	1976	482	1065	1077	1894.6	20123.8
		11938	4057	4885	3941	1126	1066	1077	2617.5	40000.8

Table 4.4: Results when installation costs for express pipes are significant

Example		Cutting Planes			LPs	B&B	LB	UB	Times (sec)	
	init	knap	cuts	cycle					Heur	Total
nw	25	25	9	0	7	1	323	323	0.0	0.1
nw.p	25	76	3	0	16	1	256	256	0.0	0.3
nw3	52	199	42	39	25	1	567	567	0.0	3.9
nw3.p	51	442	17	167	88	1	453	453	0.0	16.4
nw3.0	52	184	42	36	20	1	567	567	0.0	3.7
nw3.0.p	51	389	16	168	100	1	437	437	0.0	15.7
nw3.d1.p	51	1078	35	259	78	1	292	292	2.4	56.3
terbco	42	64	25	0	8	1	746	746	0.0	0.5
terbstar	150	4840	5024	1518	1582	81	626	678	1038.5	10002.3
		8732	10460	2883	2903	193	629	678	1464.9	20028.5
		15728	19508	5491	5084	383	632	675	2888.8	40007.1
terbstco	192	4976	1330	1085	777	17	1370	1424	4143.3	10096.8
		6278	2813	1657	1450	58	1374	1424	5438.5	20166.5
		8773	5863	2575	2375	127	1378	1424	7263.0	40060.3

Table 4.5: Results when installation costs for express pipes are high

Part II

General Integer Programs: Recognizing and Exploiting Block Structure

In Part I of this thesis we got to know three different type of problems that lead to integer programs with block structure. We showed to what extend polyhedral methods can help to solve these problems. In particular, we encountered various new classes of inequalities and realized how many, different, and complex inequalities are necessary to describe the associated polyhedra. But, we also saw how important these inequalities – individual as well as joint inequalities – are to solve problems of realistic size. We finally noticed the limits of a branch-and-cut approach when it comes to solve examples of large scale, even if we exploit all the structure that is inherited by these problems.

Part II of this thesis is devoted to the solution of general mixed integer programs, i. e., problems where a linear objective function is to be minimized subject to a system of linear inequalities with the additional requirement that part or all of the variables must be integer. The difference to Part I is that we have only this integer program at hand and we do not know anything about the application that led to this integer program. This especially means that we cannot exploit the structure that is inherited by the application if it is not recognizable from the integer programming formulation. It is therefore no wonder that one (major) part of a general mixed integer programming solver is to perform a structural analysis of the integer program to be solved. We will discuss this issue and further important ingredients of a mixed integer programming solver in Chapter 5 by introducing our implementation of such a solver.

The main focus of Part II is to investigate whether general integer programs have block structure and whether it is possible to exploit such a structure in the solution process. In Chapter 6 we develop an algorithm that allows to recognize block structure in a general mixed integer programming formulation. The remaining two chapters give two possibilities of exploiting block structure. The first, Chapter 7, deals with the dual simplex algorithm, the method is commonly used within a branch-and-cut algorithm to solve the underlying linear programs. We have seen, for instance, in Chapter 3 that a significant amount of time can be spent to solve the underlying LPs when block structure is present. One way of speeding up the solution of linear programs is to exploit parallelism and as we will see integer linear programs with block structure are good candidates in this respect. In Chapter 8 we try to exploit block structure polyhedrally. We derive a new family of inequalities that is valid for general integer programs. These inequalities may be viewed as *individual* inequalities in the sense discussed in Part I of this thesis. We will show that incorporating these inequalities into an integer programming solver indeed helps to solve mixed integer programs.

Chapter 5

Solving General Mixed Integer Programs

The solution of general mixed integer programs is one of the challenging problems in discrete optimization. The problems that can be modeled as mixed integer programs arise, for instance, in science, technology, business, and environment, and their number is tremendous. It is therefore no wonder that many solution methods and codes exist for the solution of mixed integer programs, and not just a view of them are business oriented, see Sharda [1995] for a survey on commercial linear and integer programming solvers.

In this chapter we describe the implementation of a general mixed integer programming solver, called SIP. SIP stands for Solving Integer Programs. One motivation for the development of SIP was to have an algorithmic frame that allows to incorporate and practically evaluate new achievements in the theory of integer programming. We get to know two examples in this respect in Section 5.4 and Chapter 8. SIP and many other successful integer programming solvers are branch-and-cut algorithms. We will discuss the main ingredients of such an algorithm in this chapter and we will see that some further aspects come into play that have not yet been discussed for the branch-and-cut algorithms in Part I of this thesis. In particular, we deal with preprocessing in Section 5.1, branch-and-bound issues in Section 5.2, and cut generation in Sections 5.3 and 5.4.

As test set to evaluate the features to be discussed we use the library Miplib, a collection of real-world mixed integer programming problems, see Appendix D for details on the problem data. We show for some of the features the computational effectiveness by comparing them to the default strategy used in SIP. Table 5.1 presents the results on the Miplib when using default SIP. For all our test runs we supplied a time limit of 3600 CPU seconds and a limit on the number of branch-and-bound nodes of one million. The first column gives the name of the problem, followed by the number of branch-and-bound nodes and the number of added cutting planes. In the next two columns the best dual and primal bound are presented. Column *Time* shows the CPU seconds on a Sun Enterprise 3000 with four 168 MHz UltraSPARC processors[1] and 1024 MB main memory. The gap in percentage between the best lower and upper bound, i. e., the value $100 \cdot \frac{|\text{upper bound - lower bound}|}{|\text{lower bound}|}$, can be read from the last column. The entry is positive if the problem could not be solved within the limits; a dash means that SIP was not able to find a feasible solution, in this case the problem does not contribute to the total sum of the gaps in the last line of the table.

What one would like to have at this point is a comparison with other codes. This

[1]default SIP runs sequentially, i. e., we used just one of the processors.

is, however, very difficult. People have different machines with different storage spaces, use different packages for the solution of subproblems like linear programs, and so on. Comparable codes are in particular ABACUS, developed at the University of Cologne (Thienel [1995]), bc-opt, developed at CORE (Cordier, Marchand, Laundy, and Wolsey [1997]), CPLEX, developed at Incline Village (CPLEX [1997]), MIPO, developed at Columbia University (Balas, Ceria, and Cornuéjols [1996]), and MINTO, developed at Georgia Institute of Technology (Nemhauser, Savelsbergh, and Sigismondi [1994]). CPLEX is certainly one of the most frequently used codes, and it has become customary to compare one's own developments with CPLEX. We follow this custom, but refrain from giving comparisons to all other mentioned codes.

In Table 5.2 we run the default strategy of CPLEX Version 5.0 on the Miplib. However, we want to stress that such a comparison must be handled with caution, although we have CPLEX available and can run it in the same environment. CPLEX has so many alternative options and possible parameter settings that it is often the case that the default strategy is not the best possible setting. The reason for giving this comparison here is not to compare result by result, as there are always problems where one code is better than the other and vice versa. All we want to demonstrate at this point is that SIP can compete with state-of-the-art mixed integer programming solvers, which can be seen from the summary lines in Tables 5.1 and 5.2.

5.1 Preprocessing

Preprocessing aims at eliminating redundant information from the problem formulation given by the user and simultaneously tries to strengthen the formulation by logical implications. Preprocessing can be very effective and sometimes it might not be possible to solve certain problems without a good preprocessing. This includes, for instance, Steiner tree problems, see Koch and Martin [1998], or set partitioning problems, see Borndörfer [1998]. Typically, preprocessing is applied only once at the beginning of the solution procedure, but sometimes it pays to run the preprocessing routine more often on different nodes in the branch-and-bound phase, see, for instance, Borndörfer [1998], Hoffman and Padberg [1991]. There is always the question of the break even point between the running time for preprocessing and the savings in the solution time for the whole problem. There is no unified answer to this question. It depends on the individual problem, when intensive preprocessing pays and when not. In the following we discuss the preprocessing options that are incorporated in our code SIP and ways to implement them. Most of these options are drawn from Andersen and Andersen [1995], Bixby [1994], Crowder, Johnson, and Padberg [1983], Hoffman and Padberg [1991], and Suhl and Szymanski [1994].

We assume we are given a mixed integer program in the following form:

$$
\begin{aligned}
\min\quad & c^T x \\
\text{s.t.}\quad & Ax \left\{ \begin{matrix} \leq \\ = \\ \geq \end{matrix} \right\} b \\
& l \leq x \leq u \\
& x \in \mathbb{Z}^N \times \mathbb{R}^C,
\end{aligned} \tag{5.1}
$$

where M, N, and C are finite sets with N and C disjoint, $A \in \mathbb{R}^{M \times (N \cup C)}$, $c, l, u \in \mathbb{R}^{N \cup C}$, $b \in \mathbb{R}^M$. A variable x_i, $i \in N$, is also called *binary* if $l_i = 0$ and $u_i = 1$. If some variable x_j has no upper or lower bound, we assume that $l_j = -\infty$ or $u_j = +\infty$, where ∞ might be something like 10^{10} in the implementation. Furthermore, we denote by $s_i \in \{\leq, =, \geq\}$ the sign of row i, i.e., (5.1) reads $\min\{c^T x : Ax\, s\, b,\ l \leq x \leq u,\ x \in \mathbb{Z}^N \times \mathbb{R}^C\}$. In order to avoid too many subcases in the following

Example	B & B	Cuts	Dual Bound	Primal Bound	Time	Gap %
10teams	10370	0	922	924	3600.0	0.217
air03	8	0	340160	340160	6.7	0.000
air04	1220	0	56137	56137	1532.5	0.000
air05	3588	0	26374	26374	1696.8	0.000
arki001	100776	4	7579808.299	7646059.57	3600.1	0.874
bell3a	25146	0	878430.316	878430.316	45.3	0.000
bell5	337394	1	8966406.491	8966406.491	536.7	0.000
blend2	15055	5	7.598985	7.598985	122.4	0.000
cap6000	4323	2578	-2451418.742	-1236924	3604.0	49.543
dano3mip	1	0	576.2316203	-	3710.3	-
danoint	12655	0	62.94058146	70	3600.3	11.216
dcmulti	2637	0	188182	188182	14.6	0.000
dsbmip	867	0	-305.198175	-305.198175	42.7	0.000
egout	222	0	568.1007	568.1007	0.2	0.000
enigma	8002	524	0	0	24.2	0.000
fast0507	234	0	172.2530211	177	3604.8	2.756
fiber	783	372	405935.18	405935.18	16.9	0.000
fixnet6	1669	0	3983	3983	14.6	0.000
flugpl	7976	25	1201500	1201500	4.4	0.000
gen	11	20	112313.3627	112313.3627	0.3	0.000
gesa2	209525	33	25771445.96	25783761.56	3600.0	0.048
gesa2_o	264243	0	25711931.57	25823063.47	3600.0	0.432
gesa3	5297	0	27991042.65	27991042.65	97.1	0.000
gesa3_o	74472	0	27991042.65	27991042.65	1144.7	0.000
gt2	2215	5	21166	21166	3.2	0.000
harp2	23990	15966	-73944202.17	-70801289	3600.1	4.250
khb05250	2637	0	106940226	106940226	16.3	0.000
l152lav	3209	269	4722	4722	93.8	0.000
lseu	303	164	1120	1120	1.1	0.000
misc03	699	14	3360	3360	4.1	0.000
misc06	308	0	12850.86074	12850.86074	4.2	0.000
misc07	35585	0	2810	2810	378.8	0.000
mitre	1286	3865	115155	115155	1125.8	0.000
mod008	884	371	307	307	9.8	0.000
mod010	237	3	6548	6548	5.6	0.000
mod011	6108	0	-54558535.01	-54558535.01	2791.4	0.000
modglob	1000000	0	20652263.27	20763655.71	3495.8	0.539
noswot	1000000	179	-43	-41	2270.7	4.651
nw04	1827	0	16862	16862	732.9	0.000
p0033	77	53	3089	3089	0.1	0.000
p0201	507	136	7615	7615	5.0	0.000
p0282	1345	2308	258411	258411	38.3	0.000
p0548	1610	902	8691	8691	25.3	0.000
p2756	23151	6923	3113.257351	3141	3600.2	0.891
pk1	501934	0	11	11	1581.8	0.000
pp08a	1000000	0	5446.190476	8620	2092.7	58.276
pp08aCUTS	624198	0	6970.027419	7650	3600.0	9.756
qiu	17378	0	-132.873137	-132.873137	2326.5	0.000
qnet1	17694	12	16029.69268	16029.69268	1229.9	0.000
qnet1_o	3806	3	16029.69268	16029.69268	158.6	0.000
rentacar	105	0	30356760.98	30356760.98	53.2	0.000
rgn	2505	315	82.19999924	82.19999924	9.6	0.000
rout	200371	316	1048.991823	1079.19	3600.0	2.879
set1ch	841033	0	39920.71098	67819.5	3600.0	69.886
seymour	1947	0	406.4218572	438	3601.8	7.770
stein27	4666	0	18	18	8.0	0.000
stein45	54077	0	30	30	277.7	0.000
vpm1	1000000	0	19.5	20	1892.6	2.564
vpm2	555712	0	13.75	13.75	1368.7	0.000
Total (59)	8017878	35366			77823.6	226.547

Table 5.1: `SIP` with default parameter settings

Example	B & B	Cuts	Dual Bound	Primal Bound	Time	Gap %
10teams	5505	0	917	926	3600.3	0.981
air03	8	0	340160	340160	5.6	0.000
air04	1711	0	55869.31984	56223	3601.3	0.633
air05	4987	0	26226.4271	26539	3600.5	1.192
arki001	141554	0	7579899.334	7585039.428	3600.2	0.068
bell3a	30233	0	878430.316	878430.316	34.3	0.000
bell5	1000000	0	8935997.023	9050980.167	930.2	1.287
blend2	579295	0	7.598128134	7.598985	2148.1	0.000
cap6000	4458	2135	-2451464.477	-	3601.1	-
dano3mip	5	0	576.2317752	766.5	3601.8	33.019
danoint	16038	0	63.21257952	66.5	3600.1	5.201
dcmulti	2292	0	188163.3569	188182	10.4	0.000
dsbmip	233	0	-305.198175	-305.198175	13.9	0.000
egout	100	0	568.1007	568.1007	0.1	0.000
enigma	505	0	0	0	1.0	0.000
fast0507	321	0	172.2137727	176	3606.9	2.199
fiber	243	218	405924.1392	405935.18	8.4	0.000
fixnet6	602	0	3983	3983	3.8	0.000
flugpl	3347	0	1201500	1201500	1.1	0.000
gen	4	33	112313.3627	112313.3627	0.3	0.000
gesa2	467237	0	25701301.64	25806681.14	3600.1	0.410
gesa2_o	555856	0	25689188.34	25806648.16	3600.1	0.457
gesa3	12362	0	27988245.19	27991042.65	149.9	0.000
gesa3_o	26340	0	27988244.35	27991042.65	275.2	0.000
gt2	1000000	0	20690.519	21166	900.2	2.298
harp2	145929	600	-74092004.18	-73449312	3600.2	0.867
khb05250	9608	0	106930112	106940226	41.6	0.000
l152lav	7665	0	4722	4722	117.8	0.000
lseu	204	84	1120	1120	0.7	0.000
misc03	829	0	3360	3360	4.1	0.000
misc06	246	0	12850.20368	12851.07629	2.7	0.000
misc07	31880	0	2810	2810	318.1	0.000
mitre	722	692	115155	115155	124.2	0.000
mod008	3930	18	307	307	11.5	0.000
mod010	18	60	6548	6548	2.5	0.000
mod011	15872	0	-56369797.6	-51720755.22	3600.9	8.247
modglob	1000000	0	20695096.06	20742697.19	2156.9	0.230
noswot	1000000	0	-43	-40	1835.8	6.977
nw04	1408	0	16862	16862	405.7	0.000
p0033	13	18	3089	3089	0.0	0.000
p0201	468	73	7615	7615	5.1	0.000
p0282	122	544	258411	258411	1.6	0.000
p0548	232701	712	8636.5725	8691	3600.0	0.630
p2756	31805	3753	3111.874611	3164	3600.2	1.675
pk1	450428	0	10.99880209	11	1497.8	0.000
pp08a	1000000	0	5669.52381	7820	1301.5	37.930
pp08aCUTS	991547	0	7154.172779	7450	3600.0	4.135
qiu	10422	0	-132.8843756	-132.873137	1268.8	0.000
qnet1	380	0	16029.69268	16029.69268	19.2	0.000
qnet1_o	1757	0	16028.14901	16029.69268	44.6	0.000
rentacar	51	0	30356760.98	30356760.98	20.3	0.000
rgn	3802	0	82.19999914	82.19999924	4.4	0.000
rout	220860	0	1029.78509	1133.52	3600.0	10.073
set1ch	1000000	0	39114.77014	83033.75	2966.5	112.282
seymour	1376	0	406.3527263	443	3601.2	9.019
stein27	3772	0	18	18	4.9	0.000
stein45	66891	0	30	30	231.9	0.000
vpm1	382467	0	20	20	544.5	0.000
vpm2	1000000	0	13.20436986	13.75	2280.0	4.132
Total (59)	11470409	8940			80910.4	239.811

Table 5.2: CPLEX with default parameter settings

discussion we assume without loss of generality that there are no "greater than or equal" inequalities, i.e., $s_i \in \{\leq, =\}$. We distinguish the following cases:

Empty Rows. Suppose there is some row i with no non-zero entry. If

$$s_i = \left\{ \begin{array}{c} \leq \\ = \end{array} \right\} \text{and} \left\{ \begin{array}{cc} b_i < & 0 \\ |b_i| > & 0 \end{array} \right\}$$

the problem is infeasible, otherwise row i can be removed.

Empty/Infeasible/Fixed Columns. For all columns j check the following: If

$$l_j > u_j,$$

the problem is infeasible. If

$$u_j = l_j,$$

fix column j to its lower (or upper) bound, update the right-hand side, and delete j from the problem.

Suppose some column j has no non-zero entry. If

$$\left\{ \begin{array}{cc} l_j = & -\infty \\ u_j = & \infty \end{array} \right\} \text{and} \left\{ \begin{array}{cc} c_j > & 0 \\ c_j < & 0 \end{array} \right\}$$

the problem is unbounded (or infeasible in case no feasible solution exists). Otherwise, if

$$\left\{ \begin{array}{rcr} l_j & > & -\infty \\ u_j & < & \infty \\ -l_j = u_j & = & \infty \end{array} \right\} \text{and} \left\{ \begin{array}{ccc} c_j & \geq & 0 \\ c_j & \leq & 0 \\ & & \end{array} \right\} \text{fix column } j \text{ to} \left\{ \begin{array}{c} l_j \\ u_j \\ 0 \end{array} \right\}.$$

Parallel Rows. Suppose we are given two rows $A_{i\cdot}^T x \, s_i \, b_i$ and $A_{j\cdot}^T x \, s_j \, b_j$. Row i and j are called parallel if there is some $\alpha \in \mathbb{R}$ such that $\alpha A_{i\cdot} = A_{j\cdot}$. The following situations might occur:

1. Conflicting constraints:
   (a) s_i = '=', s_j = '=', and $\alpha b_i \neq b_j$
   (b) s_i = '=', s_j = '$\leq$', and $\alpha b_i > b_j$
   (c) s_i = '$\leq$', s_j = '$\leq$', and $\alpha b_i > b_j$ ($\alpha < 0$)

   In any of these cases the problem is infeasible.
2. Redundant constraints:
   (a) s_i = '=', s_j = '=', and $\alpha b_i = b_j$
   (b) s_i = '=', s_j = '$\leq$', and $\alpha b_i \leq b_j$
   (c) s_i = '$\leq$', s_j = '$\leq$', and $\alpha b_i \leq b_j$ ($\alpha > 0$)

   In any of these cases row j is redundant.
3. Range constraints:
   (a) s_i = '$\leq$', s_j = '$\leq$', and $\alpha b_i = b_j$ ($\alpha < 0$)
   The two inequalities can be aggregated into one equation.
   (b) s_i = '$\leq$', s_j = '$\leq$', and $\alpha b_i < b_j$ ($\alpha < 0$)
   In this case both inequalities can be aggregated into one range constraint of the form $A_{i\cdot}^T x + u = b_i$ with $0 \leq u \leq b_j - \alpha b_i$.

The question remains how to find parallel rows. Tomlin and Welsh [1986] describe an efficient procedure, when the matrix A is stored columnwise, and Andersen and Andersen [1995] slightly refine this approach. The idea is to use a hash function such that rows in different baskets are not parallel. Possible hash functions are the number of non-zeros, the index of the first and/or the last index of the row, the coefficient of the first non-zero entry, etc. In practice, the baskets are rather small so that rows inside one basket can be checked pairwise.

Duality Fixing. Suppose there is some column j with $c_j \geq 0$ that satisfies $a_{ij} \geq 0$ if s_i = '$\leq$', and $a_{ij} = 0$ if s_i = '$=$' for $i \in M$. If $l_j > -\infty$, we can fix column j to its lower bound. If $l_j = -\infty$ the problem is unbounded or infeasible. The same arguments apply to some column j with $c_j \leq 0$. Suppose $a_{ij} \leq 0$ if s_i = '$\leq$', $a_{ij} = 0$ if s_i = '$=$' for $i \in M$. If $u_j < \infty$, we can fix column j to its upper bound. If $u_j = \infty$ the problem is unbounded or infeasible.

Singleton Rows. If there is some row i that contains just one non-zero entry $a_{ij} \neq 0$, for some $j \in N \cup C$ say, then we can update the bound of column j in the following way. Initially let $-\bar{l}_j = \bar{u}_j = \infty$ and set

$$\bar{u}_j = b_i/a_{ij} \quad \text{if} \quad \begin{cases} s_i = \text{`}\leq\text{'}, a_{ij} > 0 \text{ or} \\ s_i = \text{`}=\text{'} \end{cases}$$

$$\bar{l}_j = b_i/a_{ij} \quad \text{if} \quad \begin{cases} s_i = \text{`}\leq\text{'}, a_{ij} < 0 \text{ or} \\ s_i = \text{`}=\text{'} \end{cases}$$

If $\bar{u}_j < \max\{l_j, \bar{l}_j\}$ or $\bar{l}_j > \min\{u_j, \bar{u}_j\}$ the problem is infeasible. Otherwise, we update the bounds by setting $l_j = \max\{l_j, \bar{l}_j\}$ and $u_j = \min\{u_j, \bar{u}_j\}$ and remove row i. In case variable x_j is integer (binary) we round down u_j to the next integer and l_j up to the next integer. If the new bounds coincide we can also delete column j after updating the right-hand side accordingly.

Singleton Columns. Suppose there is some column j with just one non-zero entry $a_{ij} \neq 0$, for some $i \in M$ say. Let x_j be a continuous variable with no upper and lower bounds. If s_i = '$\leq$' we know after *duality fixing* has been applied that either $c_j \leq 0$ and $a_{ij} > 0$ or $c_j \geq 0$ and $a_{ij} < 0$. In both cases, there is an optimal solution satisfying row i with equality. Thus we can assume that s_i = '$=$'. Now, we delete column j and row i. After solving the reduced problem we assign to variable x_j the value

$$x_j = \frac{b_i - \sum_{k \neq j} a_{ik} x_k}{a_{ij}} \; .$$

Forcing and Dominated Rows. Here, we exploit the bounds on the variables to detect so-called forcing and dominated rows. Consider some row i and let

$$\begin{aligned} L_i &= \sum_{j \in P_i} a_{ij} l_j + \sum_{j \in N_i} a_{ij} u_j \\ U_i &= \sum_{j \in P_i} a_{ij} u_j + \sum_{j \in N_i} a_{ij} l_j \end{aligned} \tag{5.2}$$

where $P_i = \{j : a_{ij} > 0\}$ and $N_i = \{j : a_{ij} < 0\}$. Obviously, $L_i \leq \sum_{j=1}^n a_{ij} x_j \leq U_i$. The following cases might come up:

1. Infeasible row:
   (a) s_i = '=', and $L_i > b_i$ or $U_i < b_i$
   (b) s_i = '$\leq$', and $L_i > b_i$

   In these cases the problem is infeasible.
2. Forcing row:
   (a) s_i = '=', and $L_i = b_i$ or $U_i = b_i$
   (b) s_i = '$\leq$', and $L_i = b_i$

   Here, all variables in P_i can be fixed to its lower (upper) bound and all variables in N_i to its upper (lower) bound when $L_i = b_i$ ($U_i = b_i$). Row i can be deleted afterwards.
3. Redundant row:
   (a) s_i = '$\leq$', and $U_i < b_i$.

This row bound analysis can also be used to strengthen the lower and upper bounds of the variables. Compute for each variable x_j

$$\bar{u}_{ij} = \begin{cases} (b_i - L_i)/a_{ij} + l_j, & \text{if } a_{ij} > 0 \\ (b_i - U_i)/a_{ij} + l_j, & \text{if } a_{ij} < 0 \text{ and } s_i = \text{'='} \\ (L_i - U_i)/a_{ij} + l_j, & \text{if } a_{ij} < 0 \text{ and } s_i = \text{'}\leq\text{'} \end{cases}$$

$$\bar{l}_{ij} = \begin{cases} (b_i - U_i)/a_{ij} + u_j, & \text{if } a_{ij} > 0 \text{ and } s_i = \text{'='} \\ (L_i - U_i)/a_{ij} + u_j, & \text{if } a_{ij} > 0 \text{ and } s_i = \text{'}\leq\text{'} \\ (b_i - L_i)/a_{ij} + u_j, & \text{if } a_{ij} < 0. \end{cases}$$

Let $\bar{u}_j = \min_i \bar{u}_{ij}$ and $\bar{l}_j = \max_i \bar{l}_{ij}$. If $\bar{u}_j \leq u_j$ and $\bar{l}_j \geq l_j$, we speak of an *implied free variable*. The simplex method might benefit from not updating the bounds but treating variable x_j as a free variable (note, setting the bounds of j to $-\infty$ and $+\infty$ will not change the feasible region). Free variables will always be in the basis and are thus useful in finding a starting basis. For mixed integer programs however, it is in general better to update the bounds by setting $u_j = \min\{u_j, \bar{u}_j\}$ and $l_j = \max\{l_j, \bar{l}_j\}$, because the search region of the variable within an enumeration scheme is reduced. In case x_j is an integer (or binary) variable we round u_j down to the next integer and l_j up to the next integer. As an example consider the following inequality (taken from `mod015` from the `Miplib`):

$$-45x_6 - 45x_{30} - 79x_{54} - 53x_{78} - 53x_{102} - 670x_{126} \leq -443 \tag{5.3}$$

Since all variables are binary, $L_i = -945$ and $U_i = 0$. For $j = 126$ we obtain $\bar{l}_{ij} = (-443 + 945)/-670 + 1 = 0.26$. After rounding up it follows that x_{126} must be one.

Note that with these new lower and upper bounds on the variables it might pay to recompute the row bounds L_i and U_i, which again might result in tighter bounds on the variables.

Coefficient Reduction. The row bounds in (5.2) can also be used to reduce coefficients of binary variables. Consider some row i with s_i = '$\leq$' and let x_j be a binary variable with $a_{ij} \neq 0$.

$$\text{If} \begin{cases} a_{ij} < 0, U_i + a_{ij} < b_i, & \text{set} \quad a'_{ij} = b_i - U_i, \\ a_{ij} > 0, U_i - a_{ij} < b_i, & \text{set} \quad \begin{cases} a'_{ij} = U_i - b_i, \\ b_i = U_i - a_{ij}, \end{cases} \end{cases} \tag{5.4}$$

where a'_{ij} denotes the new reduced coefficient. Consider the following inequality of example p0033 from the Miplib:

$$-230x_{10} - 200x_{16} - 400x_{17} \leq -5 \tag{5.5}$$

All variables are binary, $U_i = 0$, and $L_i = -830$. We have $U_i + a_{i,10} = -230 < -5$ and we can reduce $a_{i,10}$ to $b_i - U_i = -5$. The same can be done for the other coefficients, and we obtain the inequality

$$-5x_{10} - 5x_{16} - 5x_{17} \leq -5 \tag{5.6}$$

Note that the operation of reducing coefficients to the value of the right-hand side can also be applied to integer variables if all variables in this row have negative coefficients and lower bound zero. In addition, we may compute the greatest common divisor of the coefficients and divide all coefficients and the right-hand side by this value. In case all involved variables are integer (or binary) the right-hand side can be rounded down to the next integer. In our example, the greatest common divisor is 5, and dividing by that number we obtain the set covering inequality

$$-x_{10} - x_{16} - x_{17} \leq -1. \tag{5.7}$$

Aggregation. In mixed integer programs very often equations of the form

$$a_{ij}x_j + a_{ik}x_k = b_i$$

appear for some $i \in M$, $k, j \in N \cup C$. In this case, we may replace one of the variables, x_k say, by

$$\frac{b_i - a_{ij}x_j}{a_{ik}} . \tag{5.8}$$

In case x_k is binary or integer, the substitution is only possible, if the term (5.8) is guaranteed to be binary or integer as well. If this is true or x_k is a continuous variable, we aggregate the two variables. The new bounds of variable x_j are $l_j = \max\{l_j, (b_i - a_{ik}l_k)/a_{ij}\}$ and $u_j = \min\{u_j, (b_i - a_{ik}u_k)/a_{ij}\}$ if $a_{ik}/a_{ij} < 0$, and $l_j = \max\{l_j, (b_i - a_{ik}u_k)/a_{ij}\}$ and $u_j = \min\{u_j, (b_i - a_{ik}l_k)/a_{ij}\}$ if $a_{ik}/a_{ij} > 0$.

Of course, aggregation can also be applied to equations whose support is greater than two. However, this might cause additional fill in the matrix. Hence, aggregation is usually restricted to constraints and columns with small support; in SIP we only perform the discussed case of two variables.

Disaggregation. Disaggregation of columns is to our knowledge not an issue in mixed integer programming, since this usually blows up the solution space. It is however applied in interior point algorithms for linear programs, because dense columns result in dense blocks in the Cholesky decomposition and are thus to be avoided, see Gondzio [1994].

On the other hand, disaggregation of rows is an important issue for mixed integer programs. Consider the following inequality (taken from the Miplib-problem p0282)

$$x_{85} + x_{90} + x_{95} + x_{100} + x_{217} + x_{222} + x_{227} + x_{232} - 8x_{246} \leq 0 \tag{5.9}$$

where all variables involved are binary. The inequality says that whenever one of the variables x_i with $i \in S := \{85, 90, 100, 217, 222, 227, 232\}$ is one,

x_{246} must also be one. This fact can also be expressed by replacing (5.9) by the following eight inequalities:

$$x_i - x_{246} \leq 0 \qquad \text{for all } i \in S. \tag{5.10}$$

Concerning the LP-relaxation, this formulation is tighter. Whenever any variable in S is one, x_{246} is forced to one as well, which is not guaranteed in the original formulation. On the other hand, one constraint is replaced by many (in our case 8) inequalities, which might blow up the constraint matrix. However within a cutting plane procedure this problem is not really an issue, because the inequalities in (5.10) can be generated on demand.

Probing. Probing is sometimes used in integer programming codes, see for instance Suhl and Szymanski [1994]. The idea is to set some binary variable temporary to zero or one and try to deduce further fixings from that. These implications can be expressed in inequalities as follows:

$$\begin{array}{ll} (x_j = 1 \Rightarrow x_i = \alpha) & \Rightarrow \left\{ \begin{array}{ll} x_i \geq & l_i + (\alpha - l_i)x_j \\ x_i \leq & u_i - (u_i - \alpha)x_j \end{array} \right. \\ \\ (x_j = 0 \Rightarrow x_i = \alpha) & \Rightarrow \left\{ \begin{array}{ll} x_i \geq & \alpha - (\alpha - l_i)x_j \\ x_i \leq & \alpha + (u_i - \alpha)x_j \end{array} \right. \end{array} \tag{5.11}$$

As an example, suppose we set in (5.9) variable x_{246} temporary to zero. This implies that $x_i = 0$ for all $i \in S$. Applying (5.11) we deduce the inequality

$$x_i \leq 0 + (1 - 0)x_{246} = x_{246}$$

for all $i \in S$ which is exactly (5.10).

We have implemented all the described presolve tests (except *probing*) in `SIP`. We apply the tests iteratively until all of them fail. In other words, we try to strengthen the original formulation as far as possible. Table 5.3 shows our results for the `Miplib`-problems. Columns 2 to 4 give the problem sizes (number of rows, columns, and non-zeros) of the constraint matrix after presolve. For the original data please refer to Table D.4. Columns 5 to 7 show the reductions in percentage, for example the number of rows of problem 10teams could be reduced from 230 to 210 which is a reduction of 8.7%. Note that some examples have negative percentage (misc03, p0282, p0548, p2756), because the number of rows (non-zeros) increased by applying *disaggregation* as described on page 80. Sometimes the number of rows is very high after *disaggregation* (for instance, p2756 has no less than 941 disaggregated rows and only 755 rows in the original formulation) and it seems to be self-evident not to put these constraints in the integer programming formulation, but to generate them on demand in the cutting plane phase. We have tested this alternative and surprisingly, the cutting plane alternative yields worse results than keeping all disaggregated rows in the linear program.

In summary, we see that presolve reduces the problem sizes in terms of number of rows, columns, and non-zeros by around 10%. The time spent in presolve is neglectable (25 seconds for all 59 problems). Interesting to note is also that for some problems presolve is indispensable for their solution. For example, if we turn off presolve for problem fixnet6, `SIP` is not able to solve the problem in hours.

Coming back to our initial, in general not answerable question, what is the best trade-off between work/time spent in presolve and savings in the total running time. Our answer to this question is to follow the strategy of detecting simple forms of redundancy (the presented options are basically straight-forward), but doing this fast.

Example	Absolute Values			in Percentage			Time
	Rows	Cols	NZs	Rows	Cols	NZs	
10teams	210	1600	9600	8.7	21.0	21.0	0.10
air03	122	10755	91024	1.6	0.0	0.0	0.89
air04	777	8873	69933	5.6	0.3	4.2	1.26
air05	408	7195	50762	4.2	0.0	2.6	0.49
arki001	770	973	15583	26.5	29.9	23.8	0.25
bell3a	97	110	277	21.1	17.3	20.2	0.01
bell5	85	101	251	6.6	2.9	5.6	0.01
blend2	169	319	1279	38.3	9.6	9.2	0.03
cap6000	2095	5911	17908	3.7	1.5	62.9	0.53
dano3mip	3187	13873	79625	0.5	0.0	0.0	1.70
danoint	664	521	3232	0.0	0.0	0.0	0.06
dcmulti	272	548	1297	6.2	0.0	1.4	0.03
dsbmip	1126	1826	7142	4.7	3.2	3.0	0.16
egout	40	52	104	59.2	63.1	63.1	0.01
enigma	21	100	289	0.0	0.0	0.0	0.00
fast0507	484	63001	406865	4.5	0.0	0.6	9.00
fiber	290	1055	2465	20.1	18.7	16.3	0.10
fixnet6	477	877	1754	0.2	0.1	0.1	0.03
flugpl	15	15	38	16.7	16.7	17.4	0.00
gen	622	797	2064	20.3	8.4	20.4	0.08
gesa2	1392	1224	5064	0.0	0.0	0.0	0.15
gesa2_o	1248	1224	3672	0.0	0.0	0.0	0.14
gesa3	1344	1128	4872	1.8	2.1	1.5	0.14
gesa3_o	1200	1128	3552	2.0	2.1	2.0	0.10
gt2	28	173	346	3.4	8.0	8.0	0.01
harp2	100	1373	2598	10.7	54.1	55.5	1.26
khb05250	100	1299	2598	1.0	3.8	3.8	0.05
l152lav	97	1989	9922	0.0	0.0	0.0	0.05
lseu	27	89	262	3.6	0.0	15.2	0.01
misc03	95	153	1884	1.0	4.4	8.2	0.02
misc06	666	1409	3086	18.8	22.1	47.3	0.13
misc07	223	253	8356	-5.2	2.7	3.1	0.09
mitre	1657	10724	38416	19.3	0.0	3.2	0.52
mod008	6	319	1243	0.0	0.0	0.0	0.01
mod010	146	2655	11203	0.0	0.0	0.0	0.06
mod011	2332	6895	15874	47.9	37.1	28.7	1.39
modglob	287	387	925	1.4	8.3	4.4	0.02
noswot	171	121	681	6.0	5.5	7.3	0.03
nw04	36	87482	636666	0.0	0.0	0.0	3.79
p0033	14	33	79	12.5	0.0	19.4	0.00
p0201	113	195	1677	15.0	3.0	12.8	0.03
p0282	305	202	1428	-26.6	28.4	27.4	0.05
p0548	257	477	1522	-46.0	13.0	11.0	0.06
p2756	1653	2685	9306	-118.9	2.6	-4.1	0.49
pk1	45	86	915	0.0	0.0	0.0	0.01
pp08a	136	240	480	0.0	0.0	0.0	0.01
pp08aCUTS	246	240	839	0.0	0.0	0.0	0.01
qiu	1192	840	3432	0.0	0.0	0.0	0.06
qnet1	370	1417	4294	26.4	8.0	7.1	0.07
qnet1_o	332	1417	3929	27.2	8.0	6.8	0.06
rentacar	1426	3169	22531	79.0	66.8	46.2	1.67
rgn	24	180	460	0.0	0.0	0.0	0.00
rout	290	555	2115	0.3	0.2	13.0	0.03
set1ch	446	666	1301	9.3	6.5	7.9	0.10
seymour	4827	1255	33432	2.4	8.5	0.3	0.44
stein27	118	27	378	0.0	0.0	0.0	0.00
stein45	331	45	1034	0.0	0.0	0.0	0.01
vpm1	129	189	371	44.9	50.0	50.5	0.03
vpm2	129	189	455	44.9	50.0	50.4	0.04
Total	35469	252634	1602690	7.3	10.0	12.0	25.88

Table 5.3: Presolve statistics for `Miplib`-problems

Of course, there are further and sometimes more sophisticated presolve operations possible (see, for instance, Nemhauser and Wolsey [1988], Hoffman and Padberg [1991], Borndörfer [1998]) and it remains to check whether it pays to incorporate these suggestions into our presolve algorithm.

5.2 Branch-and-Bound Strategies

In the general outline of a branch-and-cut algorithm, see Algorithm B.1 in Appendix B, there are two steps in the branch-and-bound part that leave some choices. In Step (3) of Algorithm B.1 we have to select the next problem (node) from the list of unsolved problems to work on next, and in Step (9) we must decide on how to split the problem into subproblems. Popular strategies are to branch on a variable that is closest to 0.5 and to choose a node with the worst dual bound. In this section we briefly discuss some more alternatives that are implemented in `SIP`. We will see that they sometimes outperform the mentioned standard strategy. For a comprehensive study of branch-and-bound strategies we refer to Land and Powell [1979] and Linderoth and Savelsbergh [1997] and the references therein. We assume in this section that a general mixed integer program of the form (5.1) is given.

5.2.1 Node Selection

In `SIP` there are three different strategies implemented to select the node to be processed next, see Step (3) of Algorithm B.1.

1. Best First Search (bfs).
   Here, a node is chosen with the worst dual bound, i. e., a node with lowest lower bound, since we are minimizing in (5.1). The goal is to improve the dual bound. However, if this fails early in the solution process, the branch-and-bound tree tends to grow considerably resulting in large memory requirements.

2. Depth First Search (dfs).
   This rule chooses the node that is "deepest" in the branch-and-bound tree, i. e., whose path to the root is longest. The advantages are that the tree tends to stay small, since always one of the two sons are processed next, if the node could not be fathomed. This fact also implies that the linear programs from one node to the next are very similar, usually the difference is just the change of one variable bound and thus the reoptimization goes fast. The main disadvantage is that the dual bound basically stays untouched during the solution process resulting in bad solution guarantees.

3. Best Projection.
   When selecting a node the quintessential question is, where are the good (optimal) solutions hidden in the branch-and-bound tree? In other words, is it possible to guess at some node whether it contains a better solution? Of course, this is not possible in general. But, there are some rules that evaluate the nodes according to the potential of having a better solution. One such rule is *best projection*. The earliest reference we found to this rule is Mitra [1973] who gives the credit to J. Hirst. Let $z(p)$ be the dual bound of some node p, $z(\text{root})$ the dual bound of the root node, z^*_{IP} the current best primal solution, and $s(p)$ the sum of the infeasibilities at node p, i. e., $s(p) = \sum_{i \in N} \min\{\bar{x}_i - \lfloor \bar{x}_i \rfloor, \lceil \bar{x}_i \rceil - \bar{x}_i\}$, where $\bar{x}$ is the optimal LP solution of node p and N the set of all integer variables. Let

$$\varrho(p) = z(p) + \frac{z^*_{\text{IP}} - z(\text{root})}{s(\text{root})} \cdot s(p). \tag{5.12}$$

The term $\frac{z^*_{\mathrm{IP}} - z(\mathrm{root})}{s(\mathrm{root})}$ can be viewed as a measure for the change in the objective function per unit decrease in infeasibility. The *best projection* rule selects the node that minimizes $\varrho(\cdot)$.

Tables 5.4, 5.5, and 5.6 show the results on the `Miplib`-problems when using *best-first-search, depth-first-search*, and *best projection* as node selection rules. We should note at this point that `SIP` sometimes continues at one of the sons of some node, even if the node selection rules suggests another node. The reason is that `SIP` currently contains no primal heuristic. The only way `SIP` tries to find feasible solutions is to "plunch" from time to time at some node, i. e., to dive deeper into the tree and look for feasible solutions. This strategy is very common as primal heuristics for general integer programs. Often such a heuristic is detached from the regular branch-and-bound enumeration phase, by starting at some node, alternatingly rounding some variables and solving linear programs, until all variables are fixed, the LP is infeasible, or a feasible solution has been found (see Section 4.4.2 for an example). We do not apply this rounding heuristic separately, but consider it from time to time within the global enumeration phase. The decision of whether to continue at a particular node or to choose the node suggested by the node selection rule, depends on the values $s(p)$, $\varrho(p)$ as defined in (5.12) and on the number of fractional variables. We have not turned off this option, when we performed the comparison of the three node selection rules.

Looking at the three tables we find some interesting numbers. First of all *dfs* finds by far the most feasible solutions, see Column *Sol*, where the number of times the best feasible solution could be improved is counted. This indicates that feasible solution tend to lie deep in the branch-and-bound tree. We also see that the number of simplex iterations (Column *Simplex*) per LP is on average much smaller for *dfs* (= 3.4) than using *bfs* (= 5.7) or *best projection* (= 5.5). This confirms our statement that reoptimizing a linear program is fast when just one variable bound is changed. However, *dfs* forgets to work on the dual bound. For many more difficult problems the dual bound is not improved resulting in very bad solution guarantees, compare the Column *Gap %* with the other two strategies. *Best projection* is doing a better job in this respect. It sometimes outperforms *bfs*, as for arki001 or harp2 where it finds more and better feasible solutions. However, in total *bfs* is the best strategy when comparing the total CPU time on the total gap. (Note that *best projection* even does not find a feasible solution for cap6000, and without counting this problem *bfs* is around 60% better than *best projection*.) Thus, our default strategy in `SIP` is to use best first search.

5.2.2 Variable Selection

In this section we discuss rules on how to split a problem into subproblems, if it could not be fathomed in the branch-and-bound tree, see Step (9) of Algorithm B.1. The only way to split a problem within an LP based branch-and-bound algorithm, as `SIP` is, is to branch on linear inequalities in order to keep the property of having an LP relaxation at hand. The easiest and most common inequalities are *trivial inequalities*, i. e., inequalities that split the feasible interval of a singleton variable. To be more precise, if j is some variable with a fractional value $\bar{x}_j$ in the current optimal LP solution, we obtain two subproblems, one by adding the trivial inequality $x_j \leq \lfloor \bar{x}_j \rfloor$ (called the *left subproblem* or *left son*) and one by adding the trivial inequality $x_j \geq \lceil \bar{x}_j \rceil$ (called the *right subproblem* or *right son*). This rule of branching on trivial inequalities is also called *branching on variables*, because it actually does not require to add an inequality, but only to change the bounds of variable j. Of course, one might think of branching on more complicated inequalities or even splitting the problem into more than two subproblems (see Section 6.4.4 for

Example	B & B	LPs	Simplex	Sol	Cuts	Total	Gap %
10teams	10370	10370	1819658	1	0	3600.0	0.217
air03	8	8	568	2	0	6.7	0.000
air04	1220	1220	180747	8	0	1532.5	0.000
air05	3588	3588	308477	6	0	1696.8	0.000
arki001	100776	100789	1161447	1	4	3600.1	0.874
bell3a	25146	25147	32169	4	0	45.3	0.000
bell5	337394	337396	316958	10	1	536.7	0.000
blend2	15055	15060	195660	3	5	122.4	0.000
cap6000	4323	6474	89609	1	2578	3604.0	49.543
dano3mip	1	2	157321	0	0	3710.3	-
danoint	12655	12656	2098276	5	0	3600.3	11.216
dcmulti	2637	2638	11058	22	0	14.6	0.000
dsbmip	867	868	19095	6	0	42.7	0.000
egout	222	222	216	3	0	0.2	0.000
enigma	8002	8528	103838	2	524	24.2	0.000
fast0507	234	234	52563	2	0	3604.8	2.756
fiber	783	1056	12393	9	372	16.9	0.000
fixnet6	1669	1670	12407	8	0	14.6	0.000
flugpl	7976	8051	6493	2	25	4.4	0.000
gen	11	13	337	1	20	0.3	0.000
gesa2	209525	211872	931513	15	33	3600.0	0.048
gesa2_o	264243	264244	1211802	9	0	3600.0	0.432
gesa3	5297	5298	38572	9	0	97.1	0.000
gesa3_o	74472	74473	496106	12	0	1144.7	0.000
gt2	2215	2236	8705	7	5	3.2	0.000
harp2	23990	41090	1541832	1	15966	3600.1	4.250
khb05250	2637	2637	13939	11	0	16.3	0.000
l152lav	3209	3526	54833	10	269	93.8	0.000
lseu	303	411	2445	6	164	1.1	0.000
misc03	699	746	8131	1	14	4.1	0.000
misc06	308	309	1886	5	0	4.2	0.000
misc07	35585	35585	378256	5	0	378.8	0.000
mitre	1286	2538	34802	7	3865	1125.8	0.000
mod008	884	1075	7601	7	371	9.8	0.000
mod010	237	239	2130	2	3	5.6	0.000
mod011	6108	6109	707785	16	0	2791.4	0.000
modglob	1000000	1000001	3425378	4	0	3495.8	0.539
noswot	1000000	1034126	4005738	6	179	2270.7	4.651
nw04	1827	1827	26479	3	0	732.9	0.000
p0033	77	103	355	3	53	0.1	0.000
p0201	507	609	6177	4	136	5.0	0.000
p0282	1345	2076	33468	13	2308	38.3	0.000
p0548	1610	2396	14955	11	902	25.3	0.000
p2756	23151	32416	265249	16	6923	3600.2	0.891
pk1	501934	501935	4508516	15	0	1581.8	0.000
pp08a	1000000	1000001	3950367	15	0	2092.7	58.276
pp08aCUTS	624198	624199	4560034	19	0	3600.0	9.756
qiu	17378	17379	770313	5	0	2326.5	0.000
qnet1	17694	17721	843795	16	12	1229.9	0.000
qnet1_o	3806	3811	142272	12	3	158.6	0.000
rentacar	105	106	11656	5	0	53.2	0.000
rgn	2505	2832	24239	1	315	9.6	0.000
rout	200371	200669	4161826	3	316	3600.0	2.879
set1ch	841033	841034	2261296	15	0	3600.0	69.886
seymour	1947	1947	339753	3	0	3601.8	7.770
stein27	4666	4666	13289	2	0	8.0	0.000
stein45	54077	54077	339977	3	0	277.7	0.000
vpm1	1000000	1000001	2716396	2	0	1892.6	2.564
vpm2	555712	555713	1768386	8	0	1368.7	0.000
Total (59)	8017878	8088023	46209542	403	35366	77823.6	226.547

Table 5.4: `SIP` using *best first search*

Example	B & B	LPs	Simplex	Sol	Cuts	Total	Gap %
10teams	10629	10629	2196600	8	0	3600.7	0.763
air03	31	31	852	5	0	8.9	0.000
air04	2027	2027	234285	19	0	1537.5	0.000
air05	3597	3597	223170	16	0	811.7	0.000
arki001	80122	80123	1348738	201	0	3600.7	0.942
bell3a	1000000	1000001	542291	138	0	990.1	60.799
bell5	1000000	1000002	818467	756	1	1016.1	9.446
blend2	197988	197990	2631784	122	1	1472.0	0.000
cap6000	3770	4381	458341	0	675	3600.8	-
dano3mip	1	2	157321	0	0	3830.3	-
danoint	14721	14722	2286072	2	0	3600.3	11.355
dcmulti	19571	19572	49169	85	0	78.3	0.000
dsbmip	949	950	22892	24	0	50.9	0.000
egout	197	197	228	5	0	0.1	0.000
enigma	1112	1166	15235	1	54	3.4	0.000
fast0507	560	560	69082	3	0	3601.3	2.239
fiber	1225	1515	13788	14	361	17.7	0.000
fixnet6	1725	1726	12517	28	0	13.4	0.000
flugpl	3275	3309	2326	1	25	1.4	0.000
gen	11	13	337	1	20	0.3	0.000
gesa2	291890	292370	681046	369	26	3600.0	1.073
gesa2_o	308786	308787	690344	353	0	3600.0	2.123
gesa3	277340	277902	754816	574	5	3600.0	2.927
gesa3_o	325224	325225	797242	592	0	3600.0	3.109
gt2	1000000	1001911	2057822	26	174	933.2	37.958
harp2	163246	190518	1256062	15	12250	3600.0	1.645
khb05250	2913	2913	9507	21	0	14.4	0.000
l152lav	1492	1621	22557	23	100	35.5	0.000
lseu	632	775	3832	11	222	1.7	0.000
misc03	621	650	8352	1	10	3.7	0.000
misc06	363	364	2180	12	0	4.5	0.000
misc07	37151	37151	391634	7	0	369.1	0.000
mitre	10437	12937	119663	11	3685	3607.6	0.322
mod008	8547	8738	12384	8	315	17.1	0.000
mod010	525	527	3072	3	3	8.0	0.000
mod011	7910	7911	854160	50	0	3600.6	11.957
modglob	1000000	1000001	1710911	7	0	2853.5	1.735
noswot	1000000	1004568	1769998	5	13	1606.6	13.953
nw04	1499	1499	21113	4	0	583.2	0.000
p0033	87	110	297	4	49	0.1	0.000
p0201	687	804	7302	8	203	5.7	0.000
p0282	3149	4077	36120	69	1538	40.9	0.000
p0548	2945	3555	12201	99	678	25.3	0.000
p2756	164902	169210	468995	620	985	3600.1	78.588
pk1	498407	498408	2589722	41	0	962.0	0.000
pp08a	1000000	1000001	1591203	43	0	1619.1	191.812
pp08aCUTS	1000000	1000001	2902013	52	0	3384.2	35.022
qiu	23259	23260	715386	17	0	2213.2	0.000
qnet1	27672	27875	324685	424	169	408.9	0.000
qnet1_o	17294	17383	276280	554	65	305.1	0.000
rentacar	65	66	5118	4	0	25.9	0.000
rgn	2209	2490	21125	1	219	6.8	0.000
rout	342887	343193	5216806	26	316	3600.0	14.670
set1ch	1000000	1000001	1142124	9	0	3529.4	123.840
seymour	4991	4991	419325	5	0	3600.4	7.962
stein27	4057	4057	11977	2	0	6.2	0.000
stein45	54634	54634	319027	3	0	242.9	0.000
vpm1	1000000	1000001	2084117	1	0	1557.5	27.789
vpm2	466185	466186	1615406	24	0	1017.4	0.000
Total (59)	12393517	12439184	42009419	5527	22162	85625.9	642.029

Table 5.5: `SIP` using *depth first search*

Example	B & B	LPs	Simplex	Sol	Cuts	Total	Gap %
10teams	7117	7117	1760623	6	0	3601.3	0.763
air03	8	8	568	2	0	6.7	0.000
air04	1837	1837	217726	9	0	1858.9	0.000
air05	3315	3315	297185	18	0	1834.3	0.000
arki001	89514	89525	1197250	5	7	3600.0	0.041
bell3a	18389	18390	29739	4	0	30.4	0.000
bell5	321919	321921	375767	18	1	447.7	0.000
blend2	36895	36898	347121	13	3	228.2	0.000
cap6000	4381	5895	361822	0	1772	3600.1	-
dano3mip	1	2	157321	0	0	3839.1	-
danoint	11014	11015	2100972	4	0	3600.2	6.647
dcmulti	5647	5648	15773	23	0	25.1	0.000
dsbmip	523	524	13320	7	0	29.5	0.000
egout	271	271	251	4	0	0.2	0.000
enigma	4111	4278	49195	1	167	12.4	0.000
fast0507	231	231	51341	2	0	3609.1	2.789
fiber	670	848	9342	6	299	12.7	0.000
fixnet6	1866	1867	15261	11	0	17.3	0.000
flugpl	5445	5473	4446	1	25	3.0	0.000
gen	11	13	333	1	20	0.4	0.000
gesa2	227609	229787	943408	11	35	3600.0	0.564
gesa2_o	264247	264248	1175112	9	0	3600.0	0.996
gesa3	7707	7708	59476	7	0	141.5	0.000
gesa3_o	18840	18841	78452	8	0	258.0	0.000
gt2	2014	2018	8402	6	3	3.3	0.000
harp2	43473	63975	1586741	7	13783	3600.1	0.890
khb05250	3326	3326	15286	10	0	18.9	0.000
l152lav	3278	3441	39478	7	136	69.8	0.000
lseu	330	439	2581	6	166	1.2	0.000
misc03	587	639	7093	2	16	3.5	0.000
misc06	260	261	1908	8	0	3.7	0.000
misc07	25718	25718	303230	7	0	301.4	0.000
mitre	1135	2475	34558	9	4088	1185.7	0.000
mod008	527	647	3842	3	204	4.1	0.000
mod010	155	157	1415	3	3	4.0	0.000
mod011	5313	5314	786492	13	0	3600.4	3.655
modglob	1000000	1000001	2884392	4	0	3098.5	1.161
noswot	1000000	1018128	3228959	4	160	1991.7	6.977
nw04	851	851	14642	2	0	386.4	0.000
p0033	38	51	191	1	39	0.1	0.000
p0201	461	562	5427	7	188	4.8	0.000
p0282	1334	1912	24698	13	2144	30.0	0.000
p0548	4389	6333	32130	18	1253	69.4	0.000
p2756	19061	28584	242885	23	6952	3602.2	2.384
pk1	838787	838788	7225573	20	0	2640.2	0.000
pp08a	1000000	1000001	3721267	18	0	2168.8	67.402
pp08aCUTS	692664	692665	4614783	15	0	3600.0	21.400
qiu	22753	22754	879043	8	0	2921.2	0.000
qnet1	3056	3057	87517	13	0	125.9	0.000
qnet1_o	6693	6709	201198	10	11	238.2	0.000
rentacar	143	144	14505	8	0	70.2	0.000
rgn	2347	2625	22691	1	221	8.8	0.000
rout	260447	260808	4095027	7	316	3600.0	8.716
set1ch	810890	810891	2150167	12	0	3600.0	85.411
seymour	1850	1850	333854	2	0	3604.7	7.888
stein27	4639	4639	13537	2	0	7.8	0.000
stein45	54286	54286	355329	3	0	290.3	0.000
vpm1	1000000	1000001	2834047	2	0	1929.5	19.522
vpm2	412761	412762	1360236	7	0	970.9	0.000
Total (59)	8255134	8312472	46394928	451	32012	77711.9	237.205

Table 5.6: `SIP` using *best projection*

an example), but these extensions have not yet been incorporated into `SIP`. In the following we present three variable selection rules that are implemented in `SIP`.

1. Most Infeasibility.
   This rule chooses the variable that is closest to 0.5. The heuristic reason behind this choice is that this is a variable where the least tendency can be recognized to which "side" (up or down) the variable should be rounded. The hope is that a decision on this variable has the greatest impact on the LP relaxation.

2. Pseudo-costs.
   This is a more sophisticated rule in the sense that it keeps a history of the success of the variables on which it has already been branched. To introduce this rule, which is due to Benichou, Gauthier, Girodet, Hentges, Ribiere, and Vincent [1971], we need some notation. Let $\mathcal{P}$ denote the set of all problems (nodes) except the root node that have already been solved in the solution process. Initially, this set is empty. $\mathcal{P}^+$ denotes the set of all right sons, and $\mathcal{P}^-$ the set of all left sons, where $\mathcal{P} = \mathcal{P}^+ \cup \mathcal{P}^-$. For some problem $p \in \mathcal{P}$ let

   $f(p)$ be the father of problem p.
   $v(p)$ be the variable that has been branched on to obtain problem p from the father $f(p)$.
   $x(p)$ be the optimal solution of the final linear program at node p.
   $z(p)$ be the optimal objective function value of the final linear program at node p.

   The *up pseudo-cost* of variable $j \in N$ is

   $$\Phi^+(j) = \frac{1}{|P_j^+|} \sum_{p \in P_j^+} \frac{z(p) - z(f(p))}{\lceil x_{v(p)}(f(p)) \rceil - x_{v(p)}(f(p))}, \tag{5.13}$$

   where $P_j^+ \subseteq \mathcal{P}^+$. The *down pseudo-cost* of variable $j \in N$ is

   $$\Phi^-(j) = \frac{1}{|P_j^-|} \sum_{p \in P_j^-} \frac{z(p) - z(f(p))}{x_{v(p)}(f(p)) - \lfloor x_{v(p)}(f(p)) \rfloor}, \tag{5.14}$$

   where $P_j^- \subseteq \mathcal{P}^-$. The terms $\frac{z(p)-z(f(p))}{\lceil x_{v(p)}(f(p))\rceil - x_{v(p)}(f(p))}$ and $\frac{z(p)-z(f(p))}{x_{v(p)}(f(p)) - \lfloor x_{v(p)}(f(p))\rfloor}$, respectively, measure the change in the objective function per unit decrease of infeasibility of variable j. There are many suggestions made on how to choose the sets P_j^+ and P_j^-, see Linderoth and Savelsbergh [1997] for a survey. We basically follow the suggestion of Eckstein [1994] and set $P_j^+ := \{p \in \mathcal{P}^+ : v(p) = j\}$ and $P_j^- := \{p \in \mathcal{P}^- : v(p) = j\}$, if j has already been considered as a branching variable, otherwise we use $P_j^+ := \mathcal{P}^+$ and $P_j^- := \mathcal{P}^-$. It remains to discuss how to weight the *up* and *down pseudo-costs* against each other to obtain the final *pseudo-costs* according to which the branching variable is selected. Here we set

   $$\Phi(j) = \alpha_j^+ \Phi^+(j) + \alpha_j^- \Phi^-(j), \tag{5.15}$$

   where α_j^+, α_j^- are typical positive scalars that depend on the sign of the objective function coefficient of variable j. A variable that maximizes (5.15) is chosen to be the next branching variable; ties are broken by choosing a variable that is closest to .5 . As formula (5.15) shows it takes the previously obtained success of the variables into account when deciding on the next branching variable. The weakness of this approach is that at the very beginning there is no information available, and $\Phi(\cdot)$ is almost identical for all variables. Thus, at the beginning where the branching decisions are usually the most critical the pseudo-costs take no effect. This drawback is tried to overcome in the following rule.

3. Strong Branching.
   The idea of *strong branching*, invented by CPLEX [1997] (see also Applegate, Bixby, Chvátal, and Cook [1995]), is before actually branching on some variable to test whether it indeed gives some progress. This testing is done by fixing the variable temporarily to its up and down value, i. e., to $\lceil \bar{x}_j \rceil$ and $\lfloor \bar{x}_j \rfloor$ if $\bar{x}_j$ is the fractional LP value of variable j, performing a certain fixed number of dual simplex iterations for each of the two settings, and measuring the progress in the objective function value. The testing is done, of course, not only for one variable but for a certain set of variables. Thus, the parameters of *strong branching* to be specified are the size of the candidate set, the maximum number of dual simplex iterations to be performed on each candidate variable, and a criterion according to which the candidate set is selected. In `SIP` these parameters are set as follows. We take 20% of the fractional variables, but at most 10 of them in the candidate set. The criterion according to which they are selected depends, for each variable j, on the pseudo-cost $\Phi(j)$, see (5.15), and on its infeasibility $(= \min\{\lceil \bar{x}_j \rceil - \bar{x}_j, \bar{x}_j - \lfloor \bar{x}_j \rfloor\})$. We perform for each candidate at most Δ dual simplex iterations, where Δ is initialized with 10 and is decreased the more the deeper the node lies in the branch-and-bound tree. There is no need to say that these parameter settings are all of heuristic nature and their justification are based only on experimental results.

Tables 5.7, 5.8, and 5.9 summarize our results when using these three variable selection rules. We see that branching on a *most infeasible* variable is by far the worst, measured in CPU time, in solution quality as well as in the number of branch-and-bound nodes. Using *pseudo-costs* gives much better results. The power of *pseudo-costs* becomes in particular apparent if the number of solved branch-and-bound nodes is large. In this case the function $\Phi(\cdot)$ gives a very good representation of the variables that are qualified for branching. We also observe that the time spent to compute the *pseudo-costs* is basically for free, we only spent 100 CPU seconds more time in the variable selection routine than when using *most infeasible* branching (see Column *Var. Sel.* which shows the spent CPU time). The statistics change when looking at *strong branching*. *Strong branching* is much more expensive than the other two strategies, about half of the total time is spent for selecting the variable. This comes not as a surprise, since as we have observed on page 84 the average number of dual simplex iteration per linear program is only around 6. Thus, the testing of a certain number of variables (even if it is small) in *strong branching* is relatively expensive. But, we also see that the number of branch-and-bound nodes is drastically decreased to about 40% compared to the *pseudo-costs* strategy. This decrease, however, does not completely compensate the higher running times for selecting the variables. Thus, in total *strong branching* does not outperform *pseudo-costs*, though there are examples like air04, air05 or qnet1 where *strong branching* is significantly better.

5.3 Cutting Planes from the Literature

In this section we discuss cutting planes known from the literature that are incorporated in our code `SIP`. Cutting planes for integer programs may be classified with

Example	B & B	LPs	Cuts	Var. Sel.	Total	Gap %
10teams	9726	9726	0	1.9	3600.1	0.348
air03	8	8	0	0.0	7.0	0.000
air04	3075	3075	0	3.9	3600.2	0.481
air05	5127	5127	0	6.0	3600.5	1.393
arki001	84511	84518	6	7.7	3600.0	-
bell3a	123945	123978	4	1.4	201.1	0.000
bell5	1000000	1057038	17	15.0	1572.3	4.858
blend2	75105	75111	6	3.5	608.8	0.000
cap6000	4567	4885	362	2.5	3600.6	-
dano3mip	1	2	0	0.0	3806.9	-
danoint	14401	14402	0	0.3	3600.2	9.622
dcmulti	114080	114081	0	1.3	598.0	0.000
dsbmip	7991	7992	0	0.2	391.8	0.000
egout	249	249	0	0.0	0.2	0.000
enigma	19750	21037	1282	0.3	60.7	0.000
fast0507	242	242	0	2.8	3625.6	2.759
fiber	538	859	474	0.1	17.6	0.000
fixnet6	1270	1271	0	0.0	12.3	0.000
flugpl	11445	11511	25	0.0	6.6	0.000
gen	15	17	20	0.0	0.4	0.000
gesa2	227006	234389	25	16.7	3600.0	0.968
gesa2_o	265295	265296	0	34.4	3600.0	1.402
gesa3	181901	181902	0	12.3	3600.0	0.256
gesa3_o	152302	152303	0	19.0	3600.0	8.875
gt2	1000000	1002297	174	34.0	1344.4	4.996
harp2	29480	52706	16435	8.2	3600.0	3.009
khb05250	9202	9202	0	0.1	69.8	0.000
l152lav	34519	36097	765	6.5	806.5	0.000
lseu	218	298	146	0.0	1.0	0.000
misc03	497	546	13	0.0	3.0	0.000
misc06	447	448	0	0.0	5.8	0.000
misc07	18194	18194	0	0.5	239.7	0.000
mitre	1004	2225	3589	1.0	1092.1	0.000
mod008	487	658	327	0.0	5.8	0.000
mod010	151	153	3	0.1	4.4	0.000
mod011	5276	5277	0	0.2	3600.5	5.102
modglob	1000000	1000001	0	19.2	3418.1	0.179
noswot	1000000	1043317	198	17.6	1967.3	6.977
nw04	1420	1420	0	12.6	601.2	0.000
p0033	40	56	45	0.0	0.1	0.000
p0201	1114	1279	271	0.0	10.5	0.000
p0282	8375	10424	3863	0.2	170.2	0.000
p0548	150216	199845	4140	12.1	3600.0	15.505
p2756	17205	30018	1502	8.1	3600.4	9.121
pk1	913582	913583	0	11.4	3600.0	331.527
pp08a	1000000	1000001	0	18.4	1946.1	67.640
pp08aCUTS	567341	567342	0	10.6	3600.0	15.887
qiu	19598	19599	0	0.3	3600.1	61.362
qnet1	70667	70744	21	16.7	3600.0	33.798
qnet1_o	47177	47440	120	8.2	1386.1	0.000
rentacar	137	138	0	0.0	61.9	0.000
rgn	3377	3720	348	0.1	13.3	0.000
rout	167178	167542	316	8.0	3600.0	15.592
set1ch	669968	669969	0	40.8	3600.0	114.524
seymour	2163	2163	0	0.8	3601.4	6.707
stein27	4652	4652	0	0.0	7.9	0.000
stein45	70471	70471	0	0.5	339.4	0.000
vpm1	1000000	1000001	0	15.7	2026.4	4.956
vpm2	1000000	1000001	0	18.0	3050.2	16.866
Total (59)	11116706	11320846	34497	398.9	105084.3	744.709

Table 5.7: `SIP` when branching on most infeasible variable

Example	B & B	LPs	Cuts	Var. Sel.	Total	Gap %
10teams	10370	10370	0	2.6	3600.0	0.217
air03	8	8	0	0.0	6.7	0.000
air04	1220	1220	0	1.3	1532.5	0.000
air05	3588	3588	0	3.8	1696.8	0.000
arki001	100776	100789	4	16.0	3600.1	0.874
bell3a	25146	25147	0	0.2	45.3	0.000
bell5	337394	337396	1	4.0	536.7	0.000
blend2	15055	15060	5	0.8	122.4	0.000
cap6000	4323	6474	2578	4.5	3604.0	49.543
dano3mip	1	2	0	0.0	3710.3	-
danoint	12655	12656	0	0.5	3600.3	11.216
dcmulti	2637	2638	0	0.1	14.6	0.000
dsbmip	867	868	0	0.0	42.7	0.000
egout	222	222	0	0.0	0.2	0.000
enigma	8002	8528	524	0.1	24.2	0.000
fast0507	234	234	0	3.0	3604.8	2.756
fiber	783	1056	372	0.1	16.9	0.000
fixnet6	1669	1670	0	0.1	14.6	0.000
flugpl	7976	8051	25	0.1	4.4	0.000
gen	11	13	20	0.0	0.3	0.000
gesa2	209525	211872	33	18.5	3600.0	0.048
gesa2_o	264243	264244	0	48.1	3600.0	0.432
gesa3	5297	5298	0	0.4	97.1	0.000
gesa3_o	74472	74473	0	13.0	1144.7	0.000
gt2	2215	2236	5	0.1	3.2	0.000
harp2	23990	41090	15966	7.6	3600.1	4.250
khb05250	2637	2637	0	0.1	16.3	0.000
l152lav	3209	3526	269	0.7	93.8	0.000
lseu	303	411	164	0.0	1.1	0.000
misc03	699	746	14	0.0	4.1	0.000
misc06	308	309	0	0.0	4.2	0.000
misc07	35585	35585	0	1.6	378.8	0.000
mitre	1286	2538	3865	1.5	1125.8	0.000
mod008	884	1075	371	0.0	9.8	0.000
mod010	237	239	3	0.1	5.6	0.000
mod011	6108	6109	0	0.2	2791.4	0.000
modglob	1000000	1000001	0	47.1	3495.8	0.539
noswot	1000000	1034126	179	31.7	2270.7	4.651
nw04	1827	1827	0	16.1	732.9	0.000
p0033	77	103	53	0.0	0.1	0.000
p0201	507	609	136	0.0	5.0	0.000
p0282	1345	2076	2308	0.0	38.3	0.000
p0548	1610	2396	902	0.1	25.3	0.000
p2756	23151	32416	6923	9.4	3600.2	0.891
pk1	501934	501935	0	12.3	1581.8	0.000
pp08a	1000000	1000001	0	49.4	2092.7	58.276
pp08aCUTS	624198	624199	0	27.8	3600.0	9.756
qiu	17378	17379	0	0.3	2326.5	0.000
qnet1	17694	17721	12	4.6	1229.9	0.000
qnet1_o	3806	3811	3	0.8	158.6	0.000
rentacar	105	106	0	0.0	53.2	0.000
rgn	2505	2832	315	0.0	9.6	0.000
rout	200371	200669	316	12.4	3600.0	2.879
set1ch	841033	841034	0	125.0	3600.0	69.886
seymour	1947	1947	0	1.6	3601.8	7.770
stein27	4666	4666	0	0.1	8.0	0.000
stein45	54077	54077	0	1.4	277.7	0.000
vpm1	1000000	1000001	0	20.6	1892.6	2.564
vpm2	555712	555713	0	11.0	1368.7	0.000
Total (59)	8017878	8088023	35366	501.2	77823.6	226.547

Table 5.8: SIP when branching on variables by exploiting pseudo costs

Example	B & B	LPs	Cuts	Var. Sel.	Total	Gap %
10teams	4861	9728	0	435.4	3600.3	0.326
air03	19	19	0	8.9	18.8	0.000
air04	1003	1046	0	206.3	986.8	0.000
air05	1224	1265	0	237.2	923.7	0.000
arki001	26168	26185	2	2255.6	3600.1	0.951
bell3a	33545	33547	0	7.9	77.8	0.000
bell5	698501	864662	0	2294.0	3600.0	0.019
blend2	12055	13525	0	181.3	304.7	0.000
cap6000	3779	5594	2330	661.6	3600.1	50.726
dano3mip	1	2	0	4.8	4049.9	-
danoint	8989	9238	0	636.9	3600.3	7.887
dcmulti	2199	2627	0	34.5	47.5	0.000
dsbmip	323	431	0	25.8	57.5	0.000
egout	132	192	0	0.4	0.5	0.000
enigma	3517	3742	224	8.3	20.1	0.000
fast0507	173	174	0	582.0	3616.5	1.526
fiber	240	444	296	6.4	15.1	0.000
fixnet6	724	927	0	17.6	28.1	0.000
flugpl	3260	3437	25	5.5	7.4	0.000
gen	4	6	20	0.1	0.3	0.000
gesa2	49920	53164	27	2764.8	3600.1	0.438
gesa2_o	54896	56553	0	2891.8	3600.0	0.684
gesa3	6012	6832	0	343.2	464.0	0.000
gesa3_o	20390	23197	0	983.8	1293.4	0.000
gt2	3424	3997	27	10.8	15.8	0.000
harp2	19709	33438	14270	945.0	3601.1	11.582
khb05250	2097	2516	0	31.5	46.5	0.000
l152lav	1280	1643	89	36.4	78.5	0.000
lseu	114	200	144	0.5	1.3	0.000
misc03	566	719	15	4.4	8.9	0.000
misc06	112	145	0	4.8	7.0	0.000
misc07	12091	15364	0	262.6	466.2	0.000
mitre	1096	3168	3735	450.9	1663.0	0.000
mod008	450	667	217	3.2	7.7	0.000
mod010	32	50	3	1.6	4.1	0.000
mod011	4131	4267	0	846.8	3600.2	1.596
modglob	246898	274663	0	2776.4	3600.0	0.406
noswot	355926	562953	164	2661.2	3600.0	6.977
nw04	323	427	0	342.0	554.4	0.000
p0033	12	22	26	0.0	0.1	0.000
p0201	305	437	177	3.2	7.1	0.000
p0282	801	1633	3323	18.7	50.4	0.000
p0548	589	971	707	13.0	23.4	0.000
p2756	15407	23289	5120	1607.6	3600.2	2.050
pk1	216943	259257	0	1420.3	2185.3	0.000
pp08a	364593	371902	0	2921.2	3600.0	59.176
pp08aCUTS	187974	192549	0	2653.2	3600.0	14.630
qiu	10166	11379	0	1062.6	3600.2	19.173
qnet1	3140	3442	3	149.1	315.2	0.000
qnet1_o	2314	2460	3	90.2	184.8	0.000
rentacar	51	55	0	4.6	37.7	0.000
rgn	1441	2148	256	7.2	14.6	0.000
rout	73550	85811	316	1820.2	3600.0	10.193
set1ch	127756	137835	0	3057.5	3600.1	74.954
seymour	1567	1575	0	857.1	3600.6	6.671
stein27	2326	2990	0	9.5	14.4	0.000
stein45	34525	41750	0	343.4	576.5	0.000
vpm1	418939	577515	0	2539.5	3600.0	1.839
vpm2	306912	419625	0	2565.9	3600.0	2.307
Total (59)	3349495	4157399	31519	44115.9	90178.5	274.112

Table 5.9: `SIP` using *strong branching*

regard to the question whether their derivation requires knowledge about the structure of the underlying constraint matrix. Examples of families of cutting planes that do not exploit the structure of the constraint matrix are (mixed) integer Chvátal-Gomory cuts (see Gomory [1960], Gomory [1969], Chvátal [1973], Schrijver [1980]), mixed integer rounding cuts (see Nemhauser and Wolsey [1990]), or lift-and-project cuts (see Balas, Ceria, and Cornuéjols [1993]). An alternative approach to obtain cutting planes for an integer program follows essentially the scheme to derive relaxations associated with certain substructures of the underlying constraint matrix, and tries to find valid inequalities for these relaxations. Crowder, Johnson, and Padberg [1983] pioneered this methodology by interpreting each single row of the constraint matrix as a knapsack relaxation and strengthened the integer program by adding violated knapsack inequalities. An analysis of other important relaxations of an integer program allows to incorporate odd hole and clique inequalities for the stable set polyhedron (Padberg [1973]) or flow cover inequalities for certain mixed integer models (Padberg, Roy, and Wolsey [1985], Roy and Wolsey [1987]). Further recent examples of this second approach are given in Ceria, Cordier, Marchand, and Wolsey [1998], Marchand and Wolsey [1997].

Cordier, Marchand, Laundy, and Wolsey [1997] give a nice survey on which of the mentioned cutting planes help to solve which problems from the `Miplib`. Marchand [1998] describes the merits of applying mixed integer rounding cuts. In Balas, Ceria, Cornuéjols, and Natraj [1996] it is in particular shown how useful Chvátal-Gomory cuts are if they are incorporated in the right way. Lift-and-project cuts are investigated in Balas, Ceria, and Cornuéjols [1993] and Balas, Ceria, and Cornuéjols [1996].

Currently, `SIP` separates only two of the mentioned classes of inequalities, namely weight inequalities with and without general upper bounds for the knapsack polytope. In the following we briefly recall these inequalities and discuss their separation.

Consider an instance $\mathrm{K}(N, f, F)$ of the knapsack problem, i.e., a set $N = \{1, \dots, n\}$ of items with weights $f_i, i \in N$, and a capacity F, and the polytope $P_{\mathrm{K}}(N, f, F)$ as defined in (2.4). In what follows we allow arbitrary rational weights $f_i \in \mathbb{Q}$ as well as an arbitrary capacity $F \in \mathbb{Q}$. The definitions of $\mathrm{K}(N, f, F)$ and $P_{\mathrm{K}}(N, f, F)$ are extended accordingly.

Recall from Section 2.3 the definition of an (uniform) extended weight inequality. Let $T \subseteq N$, $f(T) \leq F$, define $r := F - f(T)$ and denote by S the subset of $N \setminus T$ such that $f_i \geq r$ for all $i \in S$. The *(uniform) extended weight inequality* associated with T and some permutation $\pi_1, \dots \pi_{|S|}$ of the set S is

$$\sum_{i \in T} x_i + \sum_{i \in S} w_i x_i \leq |T|, \tag{5.16}$$

where $w_{\pi_1} = \phi(f_{\pi_1} - r)$, see (2.6) for a definition of the covering number ϕ, and, for $i = 2, \dots, |S|$, w_{π_i} is the exact lifting coefficient obtained by applying Algorithm C.2 from Appendix C, i.e., $w_{\pi_i} = |T| - \max\{\sum_{j \in T \cup \{\pi_1, \dots, \pi_{i-1}\}} w_j z_j : z \in P_{\mathrm{K}}(T \cup \{\pi_1, \dots, \pi_{i-1}\}, f_{T \cup \{\pi_1, \dots, \pi_{i-1}\}}, F - f_{\pi_i})\}$.

As noticed in Section 2.3 extended weight inequalities subsume minimal cover and $(1, k)$-configuration inequalities. The separation of minimal cover inequalities is widely discussed in the literature, because basically every general integer programming solver separates cover inequalities. The complexity of cover separation is investigated in Ferreira [1994], Klabjan, Nemhauser, and Tovey [1996], and Gu, Nemhauser, and Savelsbergh [1997], algorithmic and implementation issues are discussed among others in Crowder, Johnson, and Padberg [1983], Roy and Wolsey [1987], Zemel [1989], Hoffman and Padberg [1991], and Gu, Nemhauser, and Savelsbergh [1994]. The ideas and concepts suggested to separate cover inequalities basically carry over to extended weight inequalities. The following algorithm outlines the separation routine used in `SIP` to find violated extended weight inequalities. We call this algorithm for every row of A that contains at least one coefficient not equal to $0, \pm 1$ and whose support contains only binary variables.

Algorithm 5.3.1 *Separation algorithm of extended weight inequalities.*

Input: *A finite set* $N \subseteq \mathbb{N}$, *a vector* $a \in \mathbb{Q}^N$, *a capacity* $b \in \mathbb{Q}$ *and a vector* $\bar{x} \in [0,1]^N$.

Output: *A (with respect to* $\bar{x}$*) violated extended weight inequality that is valid for* $P_K(N, a, b)$, *or the message that none was found.*

(1) Complement variables if $a \ngeq 0$, *i. e., replace* $\bar{x}_i$ *by* $1 - \bar{x}_i$ *if* $a_i < 0$ *for* $i \in N$, *set* $b = b - \sum_{i: a_i < 0} a_i$ *and*

$$a_i = \begin{cases} -a_i, & \text{if } a_i < 0; \\ a_i & \text{if } a_i > 0. \end{cases}$$

(2) Fix all variables that are binary

$$f_i = \begin{cases} \bar{x}_i, & \text{if } \bar{x}_i \in \{0,1\}; \\ \textit{FREE}, & \text{if } i \in \operatorname{Fr}(\bar{x}) := \{j \in N : 0 < \bar{x}_j < 1\}. \end{cases}$$

(3) Determine a starting set $T \subseteq \operatorname{Fr}(\bar{x})$ *such that*

$$\sum_{i \in T} a_i x_i \leq b - \sum_{\{i:\, f_i = 1\}} a_i$$

and set $f_i = 0$ *for all* $i \in \operatorname{Fr}(\bar{x}) \setminus T$.

(4) Lift all variables in $\operatorname{Fr}(\bar{x}) \setminus T$ *exactly.*

(5) **If** *the resulting inequality is not violated –* **Stop** *(no violated inequality was found)*

(6) Lift all variables that are fixed to one exactly.

(7) Lift all variables that are fixed to zero approximately.

(8) **If** *the final inequality is violated, undo the complementing of Step (1), and return the inequality.*
Else *return with the message that no violated inequality was found.*

(9) **Stop.**

The outline of Algorithm 5.3.1 is typical for the separation of cover inequalities: fix all variables that are integer, find a cover (in our case a starting set T), and lift the remaining variables sequentially. The decisions to be made are

(a) to select the starting set T:
Here, we sort the fractional variables with respect to $\bar{x}$ in non-increasing order. We construct the set T following this order in a greedy fashion as long as $\sum_{i \in T} a_i \leq b - \sum_{\{i:\, f_i = 1\}} a_i$.

(b) to determine a lifting sequence for the variables in $N \setminus T$:
In our lifting sequence the remaining fractional variables $\operatorname{Fr}(\bar{x}) \setminus T$ come first, then all variables that are fixed to one $F_1 = \{i \in N : f_i = 1\}$, and last all variables fixed to zero $F_0 = \{i \in N : f_i = 0\}$. The variables in $\operatorname{Fr}(\bar{x}) \setminus T$ are sorted in non-increasing order with respect to $\bar{x}$, the variables in F_1 and F_0 in non-increasing order with respect to a. Denote by $\pi_1, \ldots, \pi_{|N|-|T|}$ the final lifting order.

(c) to determine the exact/approximate lifting coefficient:
The exact lifting coefficient is computed as described in Algorithm 8.5.2, where this issue is discussed in a more general setting. Algorithm 8.5.2 applied to the special situation here reads with $S = T \cup \{\pi_1, \ldots, \pi_{k-1}\}$ and the starting inequality $\sum_{i \in S} w_i x_i \leq \omega$ valid for $P_{\mathrm{K}}\left(S, a_S, b - \sum_{i \in N \setminus S} a_i f_i\right)$:

(1) **For** $l = 0, 1$ compute

$$\omega_l = \max \quad \sum_{i\in S} w_i z_i \tag{5.17}$$
$$\sum_{i\in S} a_i z_i + a_{\pi_k} \cdot l \leq b - \sum_{i\in N\setminus S} a_i f_i$$
$$z \in \{0,1\}^S.$$

(2) **If** $f_{\pi_k} = 0$ set

$$\begin{aligned} w_{\pi_k} &= \omega - \omega_1 \\ \omega &= \omega \end{aligned} \tag{5.18}$$

(3) **If** $f_{\pi_k} = 1$ set

$$\begin{aligned} w_{\pi_k} &= \omega_0 - \omega \\ \omega &= \omega_0 \end{aligned} \tag{5.19}$$

(5.17) can be solved by dynamic programming techniques in time $O(|S| \cdot (b - \sum_{i\in N\setminus S} a_i f_i))$, see Martello and Toth [1990]. Alternatively, w_l can also be determined by computing, for $\mu = 0, \ldots, \tilde{\omega}$, the integer programs

$$z(\mu) = \min \sum_{i\in S} a_i z_i \tag{5.20}$$
$$\sum_{i\in S} w_i z_i \geq \mu$$
$$z \in \{0,1\}^S,$$

and setting $w_l = \max\{\mu : z(\mu) \leq b - \sum_{i\in N\setminus S} a_i f_i - a_{\pi_k} \cdot l\}$, where $\tilde{\omega} = \omega$, if $l = 1$, and $\tilde{\omega} = \sum_{i\in S} w_i$, if $l = 0$. (5.20) can again be solved by dynamic programming techniques in time $O(|S| \cdot \tilde{\omega})$. If $f_{\pi_k} = 0$ for all k, Zemel [1989] suggested this "dual" procedure for the lifting of minimal cover inequalities in which case the procedure is polynomial, since $\tilde{\omega} \leq |N|$. We also use the "dual" procedure for the lifting in Step (4) and Step (6), since in general $\omega \leq b$ and $w_i \leq a_i$, respectively. The variables in F_0 are not exactly lifted in our implementation. Here we determine an optimal solution of the LP relaxation of (5.17) by using Dantzig's procedure (see Martello and Toth [1990] for an explanation of this procedure) and round the objective function value down to the next integer.

We finally want to point out that it is sometimes possible to determine the exact lifting coefficient without solving the integer programs in (5.17). Recall from (2.8) in Section 2.3 that the exact lifting coefficient w_{π_k} satisfies

$$\phi(a_{\pi_k} - r) - 1 \leq w_{\pi_k} \leq \phi(a_{\pi_k} - r),$$

where $r = b - \sum_{i\in T} a_i$. One can derive sufficient conditions under which the coefficient can be set to the upper bound $\phi(a_{\pi_k} - r)$. One such obvious condition that can be applied in Step (4) is the following. Suppose the items in T are sorted such that $a_1 \geq \ldots \geq a_{|T|}$ and let $h \in \{0, \ldots, |T|\}$ be such that $\sum_{i=1}^{h} a_i < a_{\pi_k} - r \leq \sum_{i=1}^{h+1} a_i$, i. e., $\phi(a_{\pi_k} - r) = h + 1$. Now, if in addition $\sum_{i=1}^{h+1} a_i \leq a_{\pi_k}$ the exact lifting coefficient is $w_{\pi_k} = h + 1$. This fact was already observed in Balas [1975] for minimal cover inequalities and was extended by Weismantel [1997] to extended weight inequalities.

We have tested `SIP` with and without the separation of extended weight inequalities. When using these inequalities we apply Algorithm 5.3.1 only at every third level in the branch-and-bound tree, and add an inequality $w^T x \leq \omega$ only if $\frac{w^T \bar{x} - \omega}{\max w_i}$ is at least 0.05. There are 16 instances in the `Miplib` containing rows that give rise to 0/1 knapsack relaxations. Table 5.10 and 5.11 show our results when using and not using extended weight inequalities. We see that the number of branch-and-bound nodes is reduced to less than 10% when weight inequalities are used. The time spent in the separation algorithm is only 653 seconds in total, which is around 5% of the total running time. Without weight inequalities we even do not find a feasible solution for cap6000 and mitre. The numbers clearly show that knapsack cuts are indispensable for the solution of integer programs that contain knapsack problems as a substructure.

Example	B & B	Cuts		Time		Gap %
		Others	Kn	Kn	Total	
cap6000	4323	0	2578	405.7	3604.0	49.543
enigma	8002	0	524	0.4	24.2	0.000
fiber	783	36	336	0.2	16.9	0.000
gen	11	10	10	0.0	0.3	0.000
harp2	23990	16	15950	68.3	3600.1	4.250
l152lav	3209	0	269	3.3	93.8	0.000
lseu	303	51	113	0.1	1.1	0.000
misc03	699	3	11	0.0	4.1	0.000
mitre	1286	1058	2807	10.4	1125.8	0.000
mod008	884	114	257	1.3	9.8	0.000
mod010	237	1	2	0.0	5.6	0.000
p0033	77	29	24	0.0	0.1	0.000
p0201	507	56	80	0.1	5.0	0.000
p0282	1345	839	1469	0.6	38.3	0.000
p0548	1610	247	655	0.8	25.3	0.000
p2756	23151	477	6446	162.3	3600.2	0.891
Total (16)	70417	2937	31531	653.5	12154.8	54.684

Table 5.10: SIP with extended weight inequalities

Example	B & B	Cuts		Time		Gap %
		Others	Kn	Kn	Total	
cap6000	4615	0	0	0.0	3600.8	-
enigma	5970	0	0	0.0	18.3	0.000
fiber	412369	1058	0	0.0	3600.0	0.188
gen	165	37	0	0.0	2.8	0.000
harp2	268925	119	0	0.0	3600.0	15.343
l152lav	2482	0	0	0.0	52.8	0.000
lseu	7914	199	0	0.0	15.8	0.000
misc03	735	6	0	0.0	4.0	0.000
mitre	11890	2489	0	0.0	3600.1	-
mod008	2362	666	0	0.0	13.4	0.000
mod010	377	22	0	0.0	8.0	0.000
p0033	61	36	0	0.0	0.1	0.000
p0201	769	170	0	0.0	6.6	0.000
p0282	1793	1970	0	0.0	33.7	0.000
p0548	50685	1203	0	0.0	638.1	0.000
p2756	85358	542	0	0.0	3600.0	26.724
Total (16)	856470	8517	0	0.0	18794.6	42.255

Table 5.11: SIP without extended weight inequalities

Cutting planes derived from knapsack relaxations can sometimes be strengthened if special ordered set inequalities $\sum_{i \in Q} x_i \leq 1$ for some $Q \subseteq N$ are available. In connection with a knapsack inequality this constraints are also called *generalized upper bound constraints (GUBs)*. The 0/1 *knapsack polytope with GUBs in standard form* is given by

$$P_{\mathrm{KGUB}} = \operatorname{conv}\{x \in \{0,1\}^N : \begin{array}{l} \sum_{i \in N} a_i x_i \leq b \\ \sum_{i \in Q_j} x_i \leq 1,\ i = 1, \ldots, q\}, \end{array} \tag{5.21}$$

where $Q_1, \ldots, Q_q$, $q \geq 1$ is a partition of N and $a \in \mathbb{Q}_+^N, b \in \mathbb{Q}_+$ (the "standard form" requires the non-negativity of the vector a). Note that the 0/1 knapsack polytope $P_{\mathrm{K}}(N, a, b)$ is the special case where $q = |N|$ (i.e., $|Q_j| = 1$ for all j). It is clear that by taking the additional SOS constraints into account stronger cutting planes may be derived. This possibility has been studied by Crowder, Johnson, and Padberg [1983], Johnson and Padberg [1981], Wolsey [1990], Nemhauser and Vance [1994], and Gu, Nemhauser, and Savelsbergh [1994]. In `SIP` we have also incorporated a prototype for separating extended weight inequalities for knapsack problems with GUBs. This algorithm works along the following lines.

Algorithm 5.3.2 *Separation algorithm of extended weight inequalities for $P_{K_{GUB}}$.*

Input: *A finite set $N \subseteq \mathbb{N}$, a vector $a \in \mathbb{Q}^N$, a capacity $b \in \mathbb{Q}$, a vector $\bar{x} \in [0,1]^N$, and an additional set of SOS inequalities.*

Output: *A (with respect to $\bar{x}$) violated extended weight inequality valid for $P_{K_{GUB}}$, or the message that none was found.*

(1) *Select a set of non-overlapping SOS constraints; let $Q_1, \ldots, Q_q$, $q \geq 1$, be the support of these constraints such that $Q_1, \ldots, Q_q$ is a partition of N.*
(2) *Transform the problem into a knapsack problem with GUBs in standard form.*
(3) *Determine a starting set $T \subseteq N$ with $\sum_{i \in T} a_i \leq b$ and $|T \cap Q_j| \leq 1$ for all $j = 1, \ldots, q$.*
(4) *Lift all variables in $N \setminus T$.*
(5) **If** *the final inequality is violated, undo the transformation in Step (2), and return the inequality.*
Else *return with the message that no violated inequality was found.*
(6) **Stop.**

Step (1) of Algorithm 5.3.2 is performed in a greedy like fashion by sorting all possible SOS constraints with respect to the ratios of the inequalities' slack to the cardinality of their intersection with the knapsack inequality. The transformation in (2) is necessary for SOS constraints Q_j with $\min\{a_i : i \in Q_j\} < 0$. For such a constraint Q_j, let $i_{\min} = \operatorname{argmin}\{a_i : i \in Q_j\}$ and set

$$a_i = \begin{cases} a_i - a_{i_{\min}}, & \text{for } i \in Q_j \setminus \{i_{\min}\}; \\ -a_{i_{\min}}, & \text{if } i = i_{\min}, \end{cases}$$

$$\bar{x}_i = \begin{cases} \bar{x}_i, & \text{if } i \neq \{i_{\min}\}; \\ 1 - \sum_{i \in Q} \bar{x}_i, & \text{if } i = i_{\min}, \end{cases}$$

and $b := b - a_{i_{\min}}$. The application of this transformation guarantees that $a \geq 0$.

Step (3) is performed in the same way as Step (3) of Algorithm 5.3.1 with the additional requirement that, for each SOS constraint, at most one variable is in the starting set T. The lifting sequence is determined such that all variables in Q_j $(j = 1, \ldots, q)$ are lifted consecutively. The SOS constraints themselves are sorted with respect to the values $\max\{\bar{x}_i : i \in Q_j\}$, $j = 1, \ldots, q$, in non-increasing order. Each SOS set Q_j is ordered according to the weights a_i, $i \in Q_j$, in non-increasing succession. Each of the lifting coefficients is determined approximately by solving the associated LP relaxation of (C.2) applied to the constraint system $Ax \leq b$ in (5.21). The lifting coefficient is the optimal objective function value rounded down to the next integer. The LP relaxation is solved by using an algorithm suggested by Johnson and Padberg [1981] which is an extension of Dantzig's procedure for the solution of the LP relaxation of the 0/1 knapsack problem.

Tables 5.12 and 5.13 show our computational results when applying or not applying Algorithm 5.3.2. We observe some, but no significant improvement when SOS constraints are taken into account. For p2756 we obtain a slightly better quality, for some other instances we obtain slightly better running times. It is worth to note that the number of branch-and-bound notes is considerably reduced. We also see that a substantial amount of time (about 25%) is spent in the separation algorithm. As noted, Algorithm 5.3.2 is currently a prototype and leaves space for improvements along the lines of Algorithm 5.3.1, for instance, by fixing certain variables, investigating different lifting sequences, and performing the lifting exactly for some of the variables. Nevertheless, our current implementation already shows the usefulness of separating knapsack inequalities with GUBs. We do obtain improvements, although they are small.

Example	B & B	Cuts		Time		Gap %
		Others	GUB Kn	GUB Kn	Total	
cap6000	4323	2578	0	1915.5	3604.0	49.543
enigma	8002	524	0	0.4	24.2	0.000
fiber	783	372	0	0.1	16.9	0.000
l152lav	3209	269	0	9.0	93.8	0.000
lseu	552	277	20	0.1	2.4	0.000
mitre	1286	3674	191	541.4	1125.8	0.000
mod010	237	3	0	0.0	5.6	0.000
p0033	57	24	8	0.0	0.1	0.000
p0201	507	93	43	0.2	5.0	0.000
p0282	1345	1477	831	1.6	38.3	0.000
p0548	1610	720	182	1.2	25.3	0.000
p2756	23151	6728	195	176.6	3600.2	0.891
Total (12)	45062	16739	1470	2646.1	8541.6	50.434

Table 5.12: SIP with GUB knapsack inequalities

Example	B & B	Cuts		Time		Gap %
		Others	GUB Kn	GUB Kn	Total	
cap6000	8632	4867	0	0.0	3602.8	49.542
enigma	8002	524	0	0.0	23.7	0.000
fiber	783	372	0	0.0	16.8	0.000
l152lav	3209	269	0	0.0	85.2	0.000
lseu	378	178	0	0.0	1.3	0.000
mitre	3131	5310	0	0.0	1200.3	0.000
mod010	237	3	0	0.0	5.7	0.000
p0033	53	39	0	0.0	0.1	0.000
p0201	521	119	0	0.0	5.0	0.000
p0282	1861	2330	0	0.0	49.2	0.000
p0548	3039	1071	0	0.0	49.1	0.000
p2756	42788	4662	0	0.0	3600.2	2.426
Total (12)	72634	19744	0	0.0	8639.3	51.968

Table 5.13: SIP without GUB knapsack inequalities

5.4 Mixed Integer Weight Inequalities

In this section we present a new family of inequalities for a general mixed integer knapsack polyhedra. We discuss the value of this family both from a theoretical and computational point of view. We deal with the polyhedron associated with the feasible solutions of a general mixed integer knapsack problem in which all variables may have arbitrary, but finite bounds. In Section 5.4.1 we introduce our model and the family of mixed integer weight inequalities that turn out to be valid for this model. Section 5.4.2 deals with an analysis of mixed integer weight inequalities in special cases. We present a family of knapsack polyhedra for which essentially mixed integer weight inequalities are sufficient to describe the associated polyhedron. Computational experiments with this family of inequalities are reported in Section 5.4.3.

5.4.1 The Family of Mixed Integer Weight Inequalities

Let N, Q be mutually disjoint finite subsets of $\mathbb{N}$. For a given vectors $a \in \mathbb{N}^{N \cup Q}$, $u \in \mathbb{N}^{N \cup Q}$, and a number $\alpha \in \mathbb{N} \setminus \{0\}$, we investigate

$$P_{\text{MIK}}\,(N, Q, a, \alpha, u) := \operatorname{conv}\{\, x \in \mathbb{Z}^N \times \mathbb{R}^Q : \sum_{i \in N \cup Q} a_i x_i \le \alpha,\; 0 \le x \le u \,\}, \tag{5.22}$$

where we abbreviate $P_{\text{MIK}}\,(N, Q, a, \alpha, u)$ by P_{MIK} if there is no way of confusion. Let us denote by $\mathcal{F}_{\text{MIK}}\,(N, Q, a, \alpha, u)$, or $\mathcal{F}_{\text{MIK}}$ for short, the set of all vectors $x \in P_{\text{MIK}}\,(N, Q, a, \alpha, u)$ with x_i integer for all $i \in N$.

In this section we study the polyhedron P_{MIK}. There is one elementary family of inequalities that is valid for P_{MIK} that under certain assumptions suffices to describe P_{MIK}. We call this family *mixed integer weight inequalities.*

Definition 5.4.1 *For $T \subseteq N \cup Q$ such that $\sum_{i \in T} a_i u_i < \alpha$, we denote by*

$$r(T) := \alpha - \sum_{i \in T} a_i u_i$$

the residual knapsack capacity *of the feasible solution $x \in \mathcal{F}_{MIK}$ with $x_i := u_i$ for all $i \in T$ and $x_i := 0$, otherwise. The* mixed integer weight inequality with respect to T *is the inequality*

$$\sum_{i \in T} a_i x_i + \sum_{i \in N \setminus T} (a_i - r(T))^+ x_i \le \alpha - r(T),$$

where $v^+ := \max\{0, v\}$ for $v \in \mathbb{R}$.

The name *mixed integer weight inequality* reflects that the coefficients of the variables in T equal their original weights. The coefficients of the (rational) variables in $Q \setminus T$ are zero. Likewise, the coefficient of an (integer) variable i in $N \setminus T$ is zero, if the weight a_i is smaller than $r(T)$. Otherwise, it is the weight a_i reduced by the value $r(T) > 0$. Note that the mixed integer weight inequalities coincide with the weight inequalities for the 0/1 knapsack polytope by Weismantel [1997], see (2.5) on page 12.

Example 5.4.2 *Consider $N = \{1, \dots, 6\}$, $Q = \{7, 8\}$, and the convex hull of tuples of vectors $x \in \{0,1\}^N \times [0,1]^Q$ that satisfy*

$$x_1 + x_2 + x_3 + x_4 + 3x_5 + 4x_6 + 2x_7 + 3x_8 \le 4$$

Setting $T := \{2, 3, 4\}$, we obtain that $r(T) = 1$. Then $a_1 - r(T) = 0$, $a_5 - r(T) = 2$ and $a_6 - r(T) = 3$. The mixed integer weight inequality associated with T reads

$$x_2 + x_3 + x_4 + 2x_5 + 3x_6 \le 3.$$

For $T := \{1, 7\}$, the mixed integer weight inequality associated with T reads

$$x_1 + 2x_7 + 2x_5 + 3x_6 \le 3.$$

Mixed integer weight inequalities are valid for the mixed integer knapsack polyhedron P_{MIK}.

Proposition 5.4.3 *For $T \subseteq N$ such that $\sum_{i \in T} a_i u_i < \alpha$, the mixed integer weight inequality with respect to T is valid for P_{MIK}.*

Proof. Let $x \in \mathcal{F}_{\mathrm{MIK}}$. If $\sum_{i \in N \setminus T:\, a_i > r(T)} x_i = 0$, then by definition,

$$\sum_{i \in T} a_i x_i + \sum_{i \in N \setminus T} (a_i - r(T))^+ x_i \leq \alpha - r(T).$$

Otherwise, $\sum_{i \in N \setminus T:\, a_i > r(T)} x_i \geq 1$ holds. We obtain

$$\begin{array}{l} \sum_{i \in T} a_i x_i + \sum_{i \in N \setminus T} (a_i - r(T))^+ x_i \qquad \leq \\ \sum_{i \in N \cup Q} a_i x_i - r(T) \sum_{i \in N \setminus T:\, a_i > r(T)} x_i \quad \leq \\ \alpha - r(T). \end{array}$$

This proves the statement. □

The family of mixed integer weight inequalities subsume various preprocessing operations that software packages have sometimes incorporated for tightening integer programming formulations.

One such example is the following coefficient-reduction operation. For $u, \alpha \in \mathbb{N}$, $u < \alpha$ consider the feasible set

$$\mathcal{I} := \{x \in \mathbb{Z}^2 : \, 0 \leq x_1 \leq u, \, 0 \leq x_2 \leq 1, \, x_1 + \alpha x_2 \leq \alpha\}.$$

It is easy to see that this feasible set has an equivalent formulation as

$$\mathcal{I} = \{x \in \mathbb{R}^2 \, : \, 0 \leq x_1 \leq u, \, 0 \leq x_2 \leq 1, \, x_1 + u x_2 \leq u\}.$$

The inequality $x_1 + u x_2 \leq u$ is a mixed integer weight inequality (with respect to $T = \{1\}$) that is valid for $\operatorname{conv}(\mathcal{I})$. In fact, this operation of tightening coefficients generalizes to the following situation.

Consider the constraint

$$\sum_{i \in R} x_i + \sum_{j \in F} \alpha x_j \leq \alpha, \tag{5.23}$$

where F is a subset of $0/1$ variables, R is a subset of integer variables, each variable $i \in R$ having an upper bound u_i and $\sum_{i \in R} u_i < \alpha$. The constraint (5.23) may be replaced by

$$\sum_{i \in R} x_i + \sum_{j \in F} (\sum_{i \in R} u_i) x_j \leq \sum_{i \in R} u_i. \tag{5.24}$$

The inequality (5.24) is a mixed integer weight inequality for

$$P := \operatorname{conv}\{x \in \mathbb{Z}^R \times \{0,1\}^F \, : \, x \text{ satisfies (5.23)}\}.$$

In fact, P may be described as

$$P = \operatorname{conv}\{x \in \mathbb{Z}^R \times \{0,1\}^F \, : \, x \text{ satisfies (5.24)}\}.$$

Investigating the latter formulation, one recognizes that all the facets of P are induced by the family of mixed integer weight inequalities,

$$x_i + \sum_{j \in F} u_i x_j \leq u_i \text{ for all } i \in R. \tag{5.25}$$

This system of inequalities together with lower and upper bound constraints on the variables describes P.

A special case of this latter preprocessing operation may also be found under the name *probing*, see page 81.

5.4.2 A Family of Mixed Integer Knapsack Polyhedra

Having introduced mixed integer weight inequalities for a general mixed integer knapsack problem, we indicate in this and the subsequent section that these inequalities are useful, at least in special cases. This section is devoted to this question from a more theoretical point of view. We demonstrate that there is a family of general mixed integer knapsack problems for which mixed integer weight inequalities are needed in order to describe the associated polyhedron. This family of problems is defined as follows.

Let N_1, N_2, Q be mutually disjoint finite subsets of $\mathbb{N}$. We study the mixed integer knapsack polyhedron $P_{\text{MIK}}(N_1 \cup N_2, Q, a, \alpha, u)$ defined in (5.22), where $a_i = 1$ for all $i \in N_1$ and $a_i \geq \lfloor \frac{\alpha}{2} \rfloor + 1$ for all $i \in N_2$.

Theorem 5.4.4 *The system of all mixed integer weight inequalities with respect to subset T of $N_1 \cup Q$, the set of all lower and upper bounds, and the two inequalities*

$$\begin{array}{lcl} \sum_{i \in N_2} x_i & \leq & 1 \\ \sum_{i \in N_1} x_i + \sum_{i \in N_2 \cup Q} a_i x_i & \leq & \alpha \end{array}$$

describe P_{MIK} if $a_i = 1$ for all $i \in N_1$ and $a_i \geq \lfloor \frac{\alpha}{2} \rfloor + 1$ for all $i \in N_2$.

Proof. Obviously, the system of inequalities is valid for P_{MIK}, see Proposition 5.4.3. It remains to show that it suffices to describe P_{MIK}. To see this, first notice that P_{MIK} is full dimensional. Let the inequality $c^T x \leq \gamma$ induce a facet F of P_{MIK}. We assume that $c^T x \leq \gamma$ is not a positive multiple of one of the non-negativity constraints, the upper bound constraints and the knapsack inequality.

Since $a > 0$, we have that $c_i \geq 0$ for all $i \in N_1 \cup N_2 \cup Q$. We define $T := \{i \in N_1 \cup Q : c_i > 0\}$. There are three claims that we need in order to show the statement,

1. $\sum_{i \in T} u_i < \alpha$;
2. The vector x^0 with $x_i^0 := u_i$ for all $i \in T$, $x_i^0 := 0$, otherwise is contained in F;
3. $c_i = 0$ for all $i \in N_1 \setminus T$, $c_i = 0$ for all $i \in \{j \in N_2 : a_j \leq r(T)\}$ and $c_i = 0$ for all $i \in Q \setminus T$.

Claim 1 follows by noting that if $\sum_{i \in T} u_i \geq \alpha$ would hold, then every point in $\mathcal{F}_{\text{MIK}} \cap F$ would satisfy the equation $\sum_{i \in N_1} x_i + \sum_{i \in N_2 \cup Q} a_i x_i = \alpha$. (Recall that $a_i = 1$ for all $i \in T \cap N_1$.) This contradicts our assumption that F is not induced by the original knapsack inequality.

Note that Claim 1 implies $x^0 \in \mathcal{F}_{\text{MIK}}$ and $r(T) > 0$.

We may assume that Claim 2 holds, for if not, then every integral point in F would satisfy the equation $\sum_{i \in N_2} x_i = 1$. Therefore, F must be the facet induced by this inequality.

Denoting by e_i the i-th unit vector we derive from Claims 1 and 2 that $(x^0, y^0) + e_i \in \mathcal{F}_{\text{MIK}}$ for all $i \in N_1 \setminus T$ and for all $i \in \{j \in N_2 : a_j \leq r(T)\}$. Moreover, $x^0 + e_i \in F$. Accordingly, $x^0 + \frac{1}{a_i} e_i \in \mathcal{F}_{\text{MIK}}$ and $x^0 + \frac{1}{a_i} e_i \in F$ for all $i \in Q \setminus T$. This shows Claim 3.

We define a function $f : N_2 \mapsto \mathbb{R}^+$ by setting

$$(5.26) \qquad \begin{array}{rll} f(i) := & \min & \sum_{j \in T} c_j z_j \\ & \text{s.t.} & \sum_{j \in T} z_j = a_i - r(T), \\ & & 0 \leq z_j \leq u_j, z_j \in \mathbb{Z} \text{ for all } j \in T \cap N_1 \\ & & 0 \leq z_j \leq u_j, z_j \in \mathbb{R} \text{ for all } j \in T \cap Q. \end{array}$$

Note that a solution of problem (5.26) attains the minimal value (with respect to c) by which the solution x^0 must be decreased in order to obtain a feasible solution with $x_i = 1$. We show

4. $c_i = f(i)$ for all $i \in N_2$.

To see that $c_i \leq f(i)$, let z be a solution of the program (5.26). The vector x^i defined as $x^i_j = x^0_j - z_j$ for $j \neq i$, and $x^i_i = 1$ is contained in $\mathcal{F}_{\text{MIK}}$. This implies that $c_i \leq f(i)$.

To see that $c_i \geq f(i)$ note that F is a facet. Therefore, there exists, for every $i \in N_2$, a feasible point $x \in F$ such that $x_i = 1$. We may assume that $x_j = 0$ for all $j \in N_1 \cup N_2 \cup Q$ with $c_j = 0$. Since $x^0 \in F$, $c^T(x - x^0) = 0$. Setting $z_j := x^0_j - x_j$ for all $j \in T$, we obtain that z is a feasible solution of (5.26). Taking into account that both x and x^0 are feasible solutions contained in F, we obtain $c_i = \sum_{j \in T} c_j z_j \geq f(i)$.

It follows that every x in $F \cap \mathcal{F}_{\text{MIK}}$ also satisfies the mixed integer weight inequality associated with T as an equation. Since F is a facet, the inequality $c^T x \leq \gamma$ (after appropriate scaling) must coincide with the mixed integer weight inequality with respect to T. □

In the more general case when we neglect the condition that $a_i \geq \lfloor \frac{\alpha}{2} \rfloor + 1$ for all $i \in N_2$, the associated polyhedron P_{MIK} is not necessarily described only by mixed integer weight inequalities. However, mixed integer weight inequalities are still needed in a minimal description of P_{MIK}, because they induce facets. This follows from

Proposition 5.4.5 *Let P_{MIK} be the mixed integer knapsack polyhedron defined in (5.22). A mixed integer weight inequality with respect to subsets T of $N \cup Q$ defines a facet of P_{MIK} if $r(T) > 0$, $\max_{i \in N \setminus T} a_i > r(T)$, and $a_i = 1$ for all $i \in T$.*

5.4.3 Experiments with Mixed Integer Weight Inequalities

In this section we investigate whether mixed integer weight inequalities occur in real-world models and examine to which extend they help in solving practical problems faster. We use again the library of mixed integer programs `Miplib` as our test set. To answer both questions we have incorporated mixed integer weight inequalities into `SIP`. We interpret each single inequality of the constraint system as a mixed integer knapsack problem. To meet the requirements in (5.22) all coefficients have to be positive, which can easily be obtained by complementing variables. In addition, all variables with a non-zero coefficient in the particular inequality must have finite lower and upper bounds. Moreover, we require that the support of the inequality is at least three, otherwise a complete description of the mixed integer knapsack polytope associated with this inequality is already obtained after presolve, see the discussions in Section 5.4.1. Finally, we do not consider inequalities whose coefficients are solely $0, \pm 1$, since no mixed integer weight inequalities can be found in this case. For convenience we call inequalities satisfying all these requirements *approved.* It turns out that 36 out of 59 `Miplib`-problems contain approved inequalities. For each of these 36 instances and for each approved inequality we try to derive mixed integer weight inequalities that cut off the current optimal LP solution. To do so, we must solve the following separation problem.

Problem 5.4.6 *Given $a \in \mathbb{N}^{N \cup Q}$, $\alpha \in \mathbb{N}$ and $\bar{x} \in \mathbb{Q}^{N \cup Q}$. Decide whether $\bar{x}$ satisfies all mixed integer weight inequalities of P_{MIK}. If not, find one that is violated by $\bar{x}$.*

Problem 5.4.6 turns out to be $\mathcal{NP}$-hard. Therefore, we developed a heuristic which proceeds along the following lines.

Algorithm 5.4.7 *Separation algorithm of mixed integer weight inequalities.*

1. *Sort the components of the vector* $(\bar{x} - u) \circ a$ *in increasing order, where* $\bar{x}$ *is the current optimal LP solution, and for vectors* v, w *the symbol* $v \circ w$ *denotes the vector with components* $v_i w_i$.

2. *Construct a set* $T \subseteq N$ *following the order of Step 1 in a greedy fashion as long as* $\sum_{i \in T} u_i a_i < \alpha$.

3. *Check whether the mixed integer weight inequality with respect to* T *is violated.*

We have experimented with different strategies to determine the set T. The ordering of the variables according to their contribution to the slack of the inequality (= left-hand side minus right-hand side) as outlined in Step 1 turned out to perform best. We made the following comparison. We used `SIP` with default settings of the parameters that includes the separation of mixed integer weight inequalities and compared it to the one where we explicitly turned off the separation of mixed integer weight inequalities. We tested both alternatives for all 36 `Miplib` problems that contain approved inequalities.

Example	B & B	Cuts		Time		Gap %
		Others	MIW	MIW	Total	
arki001	100776	0	4	418.0	3600.1	0.874
bell5	337394	0	1	9.3	536.7	0.000
blend2	15055	0	5	5.0	122.4	0.000
cap6000	4323	2578	0	344.4	3604.0	49.543
fiber	783	336	36	0.2	16.9	0.000
flugpl	7976	0	25	0.0	4.4	0.000
gen	11	10	10	0.0	0.3	0.000
gesa2	209525	0	33	159.4	3600.0	0.048
gesa2_o	264243	0	0	195.7	3600.0	0.432
gesa3	5297	0	0	3.9	97.1	0.000
gesa3_o	74472	0	0	56.6	1144.7	0.000
gt2	2215	0	5	0.1	3.2	0.000
harp2	23990	15950	16	97.5	3600.1	4.250
l152lav	3209	269	0	2.6	93.8	0.000
lseu	552	216	81	0.1	2.4	0.000
misc03	699	11	3	0.0	4.1	0.000
mitre	1286	2998	867	78.3	1125.8	0.000
mod008	884	257	114	0.4	9.8	0.000
mod010	237	2	1	0.0	5.6	0.000
noswot	1000000	0	179	21.9	2270.7	4.651
p0033	57	22	10	0.0	0.1	0.000
p0201	507	123	13	0.1	5.0	0.000
p0282	1345	2300	8	0.4	38.3	0.000
p0548	1610	837	65	0.9	25.3	0.000
p2756	23151	6641	282	41.1	3600.2	0.891
qnet1	17694	0	12	18.8	1229.9	0.000
qnet1_o	3806	0	3	3.3	158.6	0.000
rgn	2505	0	315	0.8	9.6	0.000
rout	200371	0	316	1.3	3600.0	2.879
vpm2	555712	0	0	14.6	1368.7	0.000
Total (30)	2859685	32550	2404	1474.8	33478.0	63.568

Table 5.14: `SIP` with mixed integer weight inequalities

Tables 5.14 and 5.15 show the results for 30 out of the 36 instances; for the remaining six examples we do not find mixed integer weight inequalities and the separation time was below 1% of the total CPU time. Note the numbers in Table 5.14 are just replications of the numbers in Table 5.1. Columns 1 of the tables gives the problem name, Column 2 the number of branch-and-bound nodes, and Column 3 counts the number of knapsack inequalities with and without GUBs, see Section 5.3. In Column 4 the number of violated mixed integer weight inequalities is presented. The next two columns depict the time spent for separating mixed integer weight inequalities and the total time. Again, we used a time limit of 3600 CPU seconds and limit the branch-and-bound nodes to one million. The last column *Gap %* shows the gap in percentage $(100 * \frac{|\text{upper bound - lower bound}|}{|\text{lower bound}|})$; 0.0 means we solve the problem within the time and branch-and-bound limit to optimality. We see that the quality of the solutions reduces from about 70% to 63% when using mixed integer weight inequalities (if we do not count cap6000, where there is no difference in time and solution quality, the quality even improves by a factor of 1.5 from 20% to 13%). Also, the solution time is slightly better. However, it turns out that not all examples improve. Significantly better are mitre, p0548, qnet1_o (the time is reduced by a factor of 2) as well as p2756 and rout, where the quality is improved by the same factor. A slightly worse result we obtain for qnet1. Interesting to note is noswot. Here, the quality of the solution can be improved significantly at the expense of an increase in time. The conclusion we draw from our computational experiences is that it pays to incorporate mixed integer weight inequalities, the reason for having this separation algorithm turned on in the default setting of SIP.

Example	B & B	Cuts		Time		Gap %
		Others	MIW	MIW	Total	
arki001	119866	0	0	0.0	3600.0	0.874
bell5	339908	0	0	0.0	506.0	0.000
blend2	15003	0	0	0.0	112.4	0.000
cap6000	4770	2850	0	0.0	3601.6	49.542
fiber	373	345	0	0.0	11.5	0.000
flugpl	7910	0	0	0.0	3.7	0.000
gen	15	10	0	0.0	0.4	0.000
gesa2	230517	0	0	0.0	3600.0	0.254
gesa2_o	292177	0	0	0.0	3600.0	0.416
gesa3	5297	0	0	0.0	94.1	0.000
gesa3_o	74472	0	0	0.0	1106.4	0.000
gt2	2432	0	0	0.0	3.4	0.000
harp2	24059	16110	0	0.0	3600.2	4.264
l152lav	3209	269	0	0.0	91.8	0.000
lseu	242	127	0	0.0	0.8	0.000
misc03	699	11	0	0.0	4.1	0.000
mitre	3916	3547	0	0.0	2084.7	0.000
mod008	768	240	0	0.0	6.6	0.000
mod010	237	2	0	0.0	5.7	0.000
noswot	1000000	0	0	0.0	1817.9	9.302
p0033	29	23	0	0.0	0.1	0.000
p0201	573	172	0	0.0	6.5	0.000
p0282	1708	2601	0	0.0	47.0	0.000
p0548	4072	1417	0	0.0	68.1	0.000
p2756	15891	7817	0	0.0	3603.8	1.589
qnet1	17456	0	0	0.0	1136.9	0.000
qnet1_o	6272	0	0	0.0	302.5	0.000
rgn	2809	0	0	0.0	3.8	0.000
rout	186181	0	0	0.0	3600.0	4.112
vpm2	555712	0	0	0.0	1353.6	0.000
Total (30)	2916573	35541	0	0.0	33973.5	70.355

Table 5.15: SIP without mixed integer weight inequalities

Chapter 6

Recognizing Block Structure

6.1 Introduction

In this chapter we investigate whether matrices arising from general integer programs have block structure (i. e., the removal of some rows decomposes the matrix into independent blocks) and whether it is possible to recognize such a structure. This immediately raises the question when do we consider a matrix to have block structure. In Part I we have seen three different integer programs where probably everybody agrees that they have block structure. But can this fact be recognized in the matrix? Look at an instance $\mathrm{MK}_w\,(N, M, f, F, c)$ of the multiple knapsack problem. There are $|M|$ individual blocks each consisting of a single (knapsack) constraints, and there are $|N|$ linking constraints. Assuming that $|N|$ dominates $|M|$ – as is the case in all real-world applications – the linking constraints clearly dominate the number of rows that belong to some block. Thus, if we just have the matrix at hand and do not know that it results from a multiple knapsack problem, would we consider the matrix to have block structure? The situation is different if we look at the Steiner tree packing problem. Consider the integer programming formulation (3.1) for some instance $\mathrm{STP}_w\,(G, \mathcal{N}, c, w)$. The number of linking constraints is $|E|$, whereas the number of rows in each block (constraints (3.1) (i)) is exponential in general. Even if we assume that the number of Steiner cut inequalities is (sub)linear in the number of edges at a particular LP solution, the number of constraints that belong to some block clearly dominate the number of linking constraints. In addition, whereas in the multiple knapsack case the number of blocks (knapsacks) is usually very small, the number of blocks (nets) might be large in the Steiner tree packing case.

Having these examples in mind one might conclude that the problem of decomposing a general integer programming matrix into blocks is hopeless, because it is a-priori not clear what to look for, i. e., in how many blocks should the matrix decompose, how big should each of the blocks be? However, when dealing with this problem we should not be governed solely by some few real-world applications. Rather we should ask the question how we want to exploit the block structure.

For example, suppose we would like to extend the idea of identifying single rows of the constraint matrix with a knapsack problem to more than one row. There are several possibilities to do that. We could consider some disjoint (knapsack) constraints that are linked by some other constraints. This results in an instance of the multiple knapsack problem and the easiest case to think of are two knapsacks. That is, we would like to decompose our matrix into two blocks. Another possibility is to try to identify (knapsack) constraints that strongly intersect with each other and to derive stronger cutting planes than in the single knapsack case by taking all intersecting knapsacks simultaneously into account. In this case we are looking for blocks that are highly "connected" and leave the number of blocks open. We will discuss this possibility in Chapter 8 in detail. To give a third example, we could also think of speeding up the solution of the underlying linear programs by exploiting parallelism. In Chapter 3 we have seen such a case where the solution of the linear programs turns out to be a serious problem. In this application, the number of blocks is determined by the number of processors that are available; the blocks are supposed to have about equal size in order to achieve a good load balancing.

The three examples show that depending on the method that exploits the block structure different block sizes and a different number of blocks are desirable. In this chapter we discuss a model that allows to specify these two parameters. More precisely, we consider the following problem that we call the *matrix decomposition problem*. Given some matrix A, some number β of *blocks* (sets of rows), and some *capacity* κ (maximum block-size); try to assign as many rows as possible to the blocks such that (i) each row is assigned to at most one block, (ii) each block contains at most κ rows, and (iii) no two rows in different blocks have a common non-zero entry in a column. The set of rows that are not assigned to any block is called the *border*.

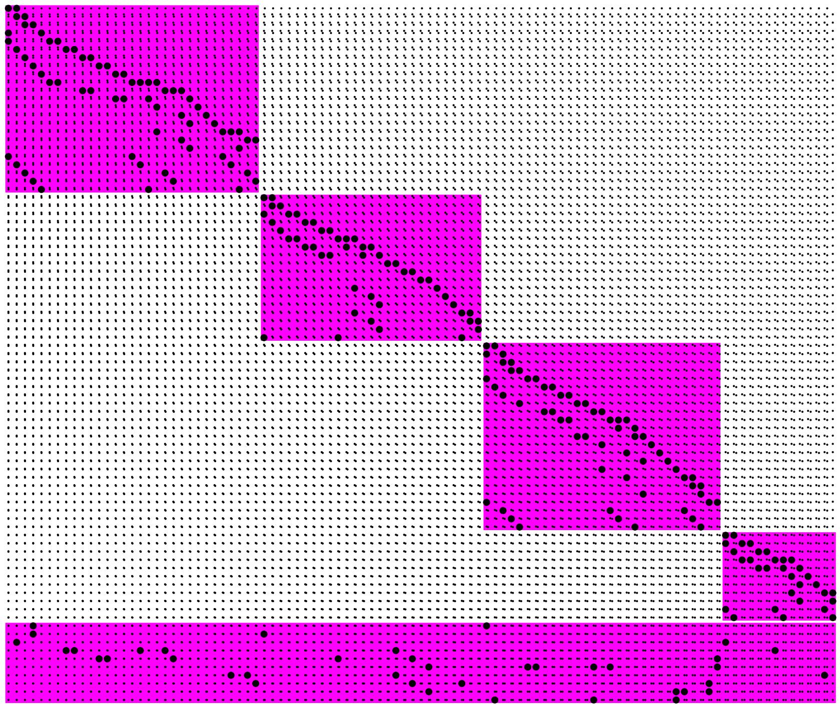
An equivalent statement of the problem in matrix terminology is as follows: Try to decompose the matrix into *bordered block diagonal form* with β blocks of capacity at most κ. The decomposition is considered the better, the smaller the number of rows in the border is; in the best case the border will be empty and the matrix decomposes into block diagonal form. Figure 6.1 (a) shows the structure of the 85×101 constraint matrix of the `Miplib`-example `bell5`, presolved by `SIP`. Figure 6.1 (b) shows an optimal decomposition of this matrix into four blocks of capacity $\lceil 85 \cdot 1.05/4 \rceil = 23$. To make the block structure of the decomposition visible, we have not only permuted the rows such that rows assigned to the same block appear consecutively, but also the columns. In this case, the matrix decomposes well, the blocks turn out to be close to square with sizes of 23×31, 18×27, 23×29, and 11×14, but in general this does not need to be the case. The border consists of only ten rows that could not be assigned to any block.

The matrix decomposition problem fits into the general context of reordering matrices to special forms. Special forms are well studied in the literature because they can be exploited by solution methods for linear equation systems, for example by LU- or Cholesky factorization, or by conjugate gradient methods. The two main points of interest are that special forms allow (i) to control fill-in (bordered block diagonal form, in particular, restricts fill-in to the blocks and the border) and (ii) independent processing of individual blocks by parallel algorithms.

Methods to obtain special forms, including (bordered) block diagonal form, are widely discussed in the literature of computational linear algebra, see, for instance, Duff, Erisman, and Reid [1986], Kumar, Grama, Gupta, and Karypis [1994], or Gallivan, Heath, Ng, Ortega, Peyton, Plemmons, Romine, Sameh, and Voigt [1990]. The matrices studied in this context mainly arise from the discretization of partial differential equations. Some newer publications deal with matrices that appear in interior point algorithms for linear programs, see Gupta [1996] and Rothberg and Hendrickson [1996], or subgradient methods for linear programs, see Ferris and Horn [1998]. Truemper [1997] developed an algorithm based on the methods in Truemper [1992] that can also be used to obtain (bordered) block diagonal form heuristically.

We develop a branch-and-cut algorithm for solving the matrix decomposition problem. Of course the expected running time of such an algorithm will neither permit its usage within a parallel LU-factorization nor within a branch-and-cut algorithm for general mixed integer programs. Our aim is rather to have a tool at

(a) bell5: Original Matrix (85 × 101)



(b) bell5: Permuted Matrix

Figure 6.1: Decomposing a matrix into bordered block diagonal form.

hand that in principle obtains an optimal bordered block diagonal form. We can then evaluate whether this special matrix structure indeed helps in solving general integer programs, and we can evaluate the success of decomposition heuristics that try to obtain (bordered) block diagonal form.

The chapter is organized as follows. In Section 6.2 we formulate the matrix decomposition problem as a 0/1 linear program and discuss connections to related combinatorial optimization problems, namely, node separation problems in graphs, the set packing, and the set covering problem. Section 6.3 is devoted to a polyhedral investigation of the matrix decomposition problem and presents (new) valid and facet-defining inequalities. In the branch-and-cut Section 6.4 we present our matrix decomposition algorithm including separation routines, primal heuristics, preprocessing, and other aspects of the implementation. We use this code in Section 6.5 to decompose matrices arising from general mixed integer programs from the `Miplib`.

6.2 An Integer Programming Formulation and Related Problems

Consider an instance MAD (A, β, κ) of the matrix decomposition problem where $A \in \mathbb{R}^{m \times n}$ is some real matrix, $\beta \in \mathbb{N}$ is the number of blocks and $\kappa \in \mathbb{N}$ is the block capacity. We introduce for each row $i = 1, \ldots, m$ and block $b = 1, \ldots, \beta$ a binary variable x_i^b that has value 1 if row i is assigned to block b and 0 otherwise. Then the matrix decomposition problem MAD (A, β, κ) can be formulated as the following 0/1 linear program.

$$
\begin{array}{lll}
\max & \displaystyle\sum_{i=1}^{m}\sum_{b=1}^{\beta} x_i^b & \\
\text{(i)} & \displaystyle\sum_{b=1}^{\beta} x_i^b \leq 1, & \text{for } i = 1, \ldots, m; \\
\text{(ii)} & \displaystyle\sum_{i=1}^{m} x_i^b \leq \kappa, & \text{for } b = 1, \ldots, \beta; \\
\text{(iii)} & x_i^b + x_j^{b'} \leq 1, & \text{for } b, b' = 1, \ldots, \beta,\ b \neq b' \text{ and} \\
 & & \text{for } i, j = 1, \ldots, m,\ i \neq j \text{ such that} \\
 & & a_{ik} \neq 0 \neq a_{jk} \text{ for some } k \in \{1, \ldots, n\}; \\
\text{(iv)} & 0 \leq x_i^b \leq 1, & \text{for } i = 1, \ldots, m,\ b = 1, \ldots, \beta; \\
\text{(v)} & x_i^b \text{ integer}, & \text{for } i = 1, \ldots, m,\ b = 1, \ldots, \beta.
\end{array}
\tag{6.1}
$$

Inequalities (i) guarantee that each row is assigned to at most one block. Constraints (ii) ensure that the number of rows assigned to a particular block b does not exceed its capacity. Finally, (iii) expresses that two rows i and j must not be assigned to different blocks if both have a non-zero entry in some common column. These three sets of inequalities plus the the trivial inequalities (iv) and the integrality constraints (v) establish a one-to-one correspondence between feasible solutions of (6.1) and block decompositions of the matrix A into β blocks of capacity κ. In the sequel we will also call a vector $x \in \mathbb{R}^{m \times \beta}$ a *block decomposition* if it is feasible for (6.1). Note that formulation (6.1) as it stands is not polynomial, since the number of variables $m\beta$ is not polynomial in the encoding length of β. However, we may assume without loss of generality $\beta \leq m$, because no more than m rows will be assigned. We also assume that the block capacity is at least one ($\kappa \geq 1$) and that we have at least two blocks ($\beta \geq 2$).

A first observation about (6.1) is that different matrices A can give rise to the same integer program or, in other words, different matrices can be decomposed in exactly the same way. In fact, such matrices form equivalence classes as can be seen by considering the *(column) intersection graph* $G(A)$ of an $m \times n$-matrix A as introduced by Padberg [1973]. $G(A)$ has the set $\{1, \ldots, n\}$ of column indices of A as its node set and there is an edge ij between two columns i and j if they have a common non-zero entry in some row. Applying this concept to the transposed matrix A^T, we obtain the *row intersection graph* $G(A^T)$ of A where two rows i and j are joined by an edge ij if and only if they have non-zero entries in a common column. But then the edges of $G(A^T)$ give rise to the inequalities (6.1) (iii) and we have that for fixed β and κ two matrices A and A' have the same row intersection graph if and only if the corresponding integer programs (6.1) are equal.

The matrix decomposition problem is related to several other combinatorial optimization problems. First, the problem can be interpreted in terms of the row intersection graph as a node separator problem. To see this, let $G(A^T) = (V, E)$ and consider some block decomposition x. The set $S := \{i \in V : \sum_{b=1}^{\beta} x_i^b = 0\}$ of rows in the border is a node separator in $G(A^T)$ such that the graph obtained by deleting all nodes in S and all its adjacent edges decomposes into at most β parts, each of cardinality at most κ. Conversely, each node separator in $G(A^T)$ with these properties gives rise to a block decomposition for MAD (A, β, κ). Various node separator problems have been studied in the literature. Lengauer [1990] gives a survey and discusses applications in VLSI design, Duff, Erisman, and Reid [1986] and Gallivan, Heath, Ng, Ortega, Peyton, Plemmons, Romine, Sameh, and Voigt [1990] emphasize heuristic methods for use in computational linear algebra. Lower bounds on the size of a node separator in a general graph are rather rare. The only results we are aware of are due to Pothen, Simon, and Liou [1990] and Helmberg, Mohar, Poljak, and Rendl [1995], who use Eigenvalue methods to derive non-trivial lower bounds on the size of a node separator for $\beta = 2$, if lower bounds on the size of the blocks are imposed.

A second connection exists to set packing, and this relationship is two-fold. On the one hand, matrix decomposition is a generalization of set packing, because feasible solutions (stable sets) of some set packing problem $\max\{\mathbb{1}^T x : Ax \leq \mathbb{1}, x \in \{0,1\}^n\}$, $A \in \{0,1\}^{m \times n}$, correspond to solutions of the matrix decomposition problem MAD $(A^T, m, 1)$ of the same objective value and vice versa. This shows that the matrix decomposition problem is $\mathcal{NP}$-hard. On the other hand, we obtain a set packing relaxation of the matrix decomposition problem by deleting the block capacity constraints (ii) from the formulation (6.1). All inequalities that are valid for this relaxation are also valid for the matrix decomposition problem and we will use some of them (namely clique- and cycle-inequalities) as cutting planes in our branch-and-cut algorithm. Note, however, that the set packing relaxation allows assignment of all rows to any single block and our computational experiments seem to indicate that these cuts are rather weak.

A close connection exists also to set covering via complementing variables. To see this we rewrite (6.1), substituting each capacity constraint (ii) by $\binom{m}{\kappa+1}$ inequalities that sum over all subsets of cardinality $\kappa + 1$ of variables $\{x_1^b, \ldots, x_m^b\}$ for some block b and each constraint in (i) by $\binom{\kappa}{2}$ inequalities that sum over all pairs of variables in $\{x_i^1, \ldots, x_i^{\beta}\}$. Replacing all variables x_i^b by $1 - y_i^b$, one obtains the set covering problem (6.2) that is stated on the following page.

This shows that the matrix decomposition problem is a (special) set covering problem. For the case of two blocks, this formulation has been used by Nicoloso and Nobili [1992] for the solution of the *matrix equipartition problem.* The matrix equipartition problem is the matrix decomposition problem for $\beta = 2$ and $\kappa = \lfloor m/2 \rfloor$, plus the additional equipartition constraint

$$\sum_{i=1}^{m} x_i^1 = \sum_{i=1}^{m} x_i^2, \quad \text{or, in complemented variables,} \quad \sum_{i=1}^{m} y_i^1 = \sum_{i=1}^{m} y_i^2,$$

that states that the two blocks of the decomposition must have equal size.

$$
(6.2)\qquad
\begin{array}{lll}
\min & \displaystyle\sum_{i=1}^{m}\sum_{b=1}^{\beta} y_i^b & \\
\text{(i)} & y_i^b + y_i^{b'} \geq 1, & \text{for } b, b' = 1, \ldots, \beta,\ b \neq b' \text{ and} \\
 & & \text{for } i = 1, \ldots, m; \\
\text{(ii)} & \displaystyle\sum_{i \in I} y_i^b \geq 1, & \text{for } b = 1, \ldots, \beta \text{ and} \\
 & & \text{for } I \subseteq \{1, \ldots, m\} \text{ with } |I| = \kappa + 1; \\
\text{(iii)} & y_i^b + y_j^{b'} \geq 1, & \text{for } b, b' = 1, \ldots, \beta,\ b \neq b' \text{ and} \\
 & & \text{for } i, j = 1, \ldots, m,\ i \neq j \text{ such that} \\
 & & a_{ik} \neq 0 \neq a_{jk} \text{ for some } k \in \{1, \ldots, n\}; \\
\text{(iv)} & 0 \leq y_i^b \leq 1, & \text{for } i = 1, \ldots, m,\ b = 1, \ldots, \beta; \\
\text{(v)} & y_i^b \text{ integer}, & \text{for } i = 1, \ldots, m,\ b = 1, \ldots, \beta.
\end{array}
$$

6.3 Polyhedral Investigations

Associated to the IP-formulation (6.1) of the matrix decomposition problem is the polytope

$$(6.3)\qquad P_{\text{MAD}}(A, \beta, \kappa) := \text{conv}\{x \in \mathbb{R}^{m \times \beta} : x \text{ satisfies (6.1) (i) to (v)}\},$$

given by the convex hull of all block decompositions. We study in this section the structure of $P_{\text{MAD}}(A, \beta, \kappa)$ to derive classes of valid and facet-defining inequalities for later use as cutting planes. We start by determining its dimension.

Proposition 6.3.1 (Dimension) *$P_{MAD}(A, \beta, \kappa)$ is full dimensional.*

Proof. The vector 0 and all unit vectors $e_i^b \in \mathbb{R}^{m \times \beta}$ are feasible, i.e., are in $P_{\text{MAD}}(A, \beta, \kappa)$, and affinely independent. □

This means that the facets of $P_{\text{MAD}}(A, \beta, \kappa)$ are uniquely determined up to a scalar factor. Two further easy observations are gathered in the following remark.

Remark 6.3.2

(i) The non-negativity inequalities $x_i^b \geq 0$ are facet-defining for all $i = 1, \ldots, m$ and all $b = 1, \ldots, \beta$.

(ii) All facet-defining inequalities $a^T x \leq \alpha$ that are not non-negativity constraints satisfy $a \geq 0$ and $\alpha > 0$.

Remark 6.3.2 (i) is proven in the same way as Theorem 6.3.1; Remark 6.3.2 (ii) is a consequence of the down monotonicity of $P_{\text{MAD}}(A, \beta, \kappa)$.

Facet-defining inequalities have another interesting property. Consider some vector $x \in \mathbb{R}^{m \times \beta}$, some permutation σ of the blocks $\{1, \ldots, \beta\}$, and define the vector $\bar{x} \in \mathbb{R}^{m \times \beta}$ by

$$\bar{x}_i^b := x_i^{\sigma(b)},$$

for $i = 1, \ldots, m, b = 1, \ldots, \beta$. We will use in the sequel the symbol $\sigma(x)$ to denote the vector $\bar{x}$ that arises from x by applying the block permutation σ. Then $\sigma(x) = \bar{x}$ is a feasible block decomposition if and only if x is. This simple observation has two consequences. First, it implies that $a^T x \leq b$ is a facet of $P_{\mathrm{MAD}}(A, \beta, \kappa)$ if and only if its block-wise permutation $\sigma(a)^T x \leq b$ is. Facets arising from each other via block permutations can thus be viewed as forming a single class that can be represented by a single member. Or, to put it in a more negative way, each facet can and will be "blown up" by block permutations to a whole set of combinatorially essentially identical conditions. Second, the objective function of the matrix decomposition problem is invariant under block permutation and thus the matrix decomposition problem is dual degenerate (has multiple optima). Both dual degeneracy and the large number of permutable facets cause difficulties in our branch-and-cut algorithm and we will have to control the number of cuts generated and to handle stalling of the objective value.

The next two subsections list the results of our polyhedral investigations in the form of valid and facet-defining inequalities. We distinguish between inequalities $a^T x \leq b$ that are *invariant under block permutations* or, equivalently, have the same coefficients $a_i^b = a_i^{b'}$ for all blocks $b \neq b'$ and row indices i, and *block-discernible inequalities* that do not have this property and distinguish different blocks. It will turn out that most of the block-discernible inequalities will be inherited from the stable set relaxation of the matrix decomposition problem, while the *block-invariant* constraints are related to an "aggregated" version of the problem. In both subsections we want to assume $\kappa \geq 2$, because otherwise the matrix decomposition problem is a (special) set packing problem.

6.3.1 Block-Discernible Inequalities

We saw in Section 6.2 that we obtain a set packing relaxation of the matrix decomposition problem by dropping the block capacity constraints (ii) from the integer program (6.1). The column intersection graph associated to the matrix $\mathrm{IP}_{(\mathrm{i}),(\mathrm{iii})}$ formed by the left-hand sides of the constraints (6.1) (i) and (iii) has the set of possible row assignments $\{1, \ldots, m\} \times \{1, \ldots, \beta\}$ as its node set. A (conflict) edge exists between two assignments (i, b) and (j, b'), if rows i and j cannot be simultaneously assigned to the blocks b and b', i. e., either if $i = j$ and $b \neq b'$ or if $i \neq j$, $b \neq b'$, and rows i and j have a common non-zero entry in some column of A. We want to call this graph the *conflict graph* associated to the matrix decomposition problem $\mathrm{MAD}(A, \beta, \kappa)$ and denote it by $G_c(A, \beta)$. In formulas: $G_c(A, \beta) = G(\mathrm{IP}_{(\mathrm{i}),(\mathrm{iii})})$. This graph allows us to interpret the inequality classes (i) and (iii) of (6.1) as *clique inequalities* of the set packing relaxation corresponding to the matrix decomposition problem as also introduced by Padberg [1973].

Theorem 6.3.3 (Clique) *Let $G_c(A, \beta) = (V, E)$ and $Q \subseteq V$. The inequality*

$$\sum_{(i,b) \in Q} x_i^b \leq 1$$

is valid for $P_{MAD}(A, \beta, \kappa)$ if and only if Q is a clique in $G_c(A, \beta)$. It is facet-defining if and only if Q is a maximal clique in $G_c(A, \beta)$.

Proof. The validity part is obvious. It remains to show that it is facet-defining if and only if Q is a maximal clique.

Suppose first that Q is not maximal but contained in a larger clique Q'. But then $\sum_{(i,b) \in Q} x_i^b \leq 1$ is the sum of the inequality $\sum_{(i,b) \in Q'} x_i^b \leq 1$ and the non-negativity constraints $x_i^b \geq 0$ for $(i, b) \in Q' \setminus Q$ and cannot be facet-defining.

Assume now that Q is maximal. We will construct a set of $m\beta$ affinely independent block decompositions for which the inequality is tight. $|Q|$ such affinely independent vectors are the unit vectors e_i^b with $(i,b) \in Q$. For each other assignment $(j,b') \notin Q$ there exists some assignment (i,b) in Q that is not in conflict with (j,b'), since Q is a maximal clique. Thus, the vector $e_j^{b'} + e_i^b$ is the incidence vector of a feasible block decomposition for which the inequality is tight. (Note that we assumed $\kappa \geq 2$ at the beginning of this section for the case $b = b'$.) The resulting $m\beta - |Q|$ characteristic vectors obtained in this way plus the $|Q|$ vectors constructed in the beginning are affinely independent. □

In the spirit of Theorem 6.3.3, (6.1) (i) and (iii) are both clique inequalities and do not represent two different types of inequalities. The separation problem for clique inequalities is a maximum-weight clique problem and thus $\mathcal{NP}$-hard, see Garey and Johnson [1979]. But some subclasses can be separated efficiently. One such class that we use in our implementation are the *two-partition inequalities*

$$\sum_{b \in B} x_i^b + \sum_{b' \notin B} x_j^{b'} \leq 1,$$

that are defined for all sets of blocks $B \subseteq \{1, \ldots, \beta\}$ and all pairs of non-disjoint rows i, j. Polynomial separation of this class is by inspection: Given i and j, we examine for each block b the variables x_i^b and x_j^b. If $x_i^b > x_j^b$, we add b to the set B, otherwise to its complement. Note that for the case of two blocks ($\beta = 2$), the two-partition inequalities are exactly the inequalities (6.1) (i) and (iii) and, moreover, these are already all clique inequalities. In particular, separation of clique inequalities is polynomial for $\beta = 2$. In general, maximal cliques in $G_c(A,\beta)$ are of the form $\{(i_1,b_1), \ldots, (i_\beta,b_\beta)\}$, where the blocks $b_k, k = 1, \ldots, \beta$ are mutually different and the set of rows $\{i_1, \ldots, i_\beta\}$ forms a clique in $G(A^T)$. Thus all maximal cliques in $G_c(A,\beta)$ are of size β.

Another class inherited from the set packing relaxation are the *cycle inequalities.*

Theorem 6.3.4 (Odd Cycle) *If C is an odd cycle in $G_c(A,\beta)$, then the cycle inequality*

$$\sum_{(i,b) \in C} x_i^b \leq \lfloor |C|/2 \rfloor$$

is valid for $P_{MAD}(A,\beta,\kappa)$.

Analogously to the set packing case, see again Padberg [1973], the odd cycle inequality is facet-defining for its support if C is an odd hole (has no chords) and $|C|/2 \leq \kappa$. These conditions are, however, not necessary. Cycle inequalities can be separated in polynomial time using the algorithm of Lemma 9.1.11 in Grötschel, Lovász, and Schrijver [1988].

Along the same lines as for the clique and cycle inequalities, the matrix decomposition polytope clearly also inherits all other packing inequalities. But not only set packing, also set covering inequalities for (6.2) can be applied (note that complementing variables preserves validity and dimension of the induced face), see Nobili and Sassano [1989]. We do, however, not use any of them for our computations.

We close this section investigating the *block capacity constraints* (6.1) (ii) which are not inherited from the set packing polytope or the set covering polytope.

Theorem 6.3.5 (Block Capacity) *The block capacity constraint*

$$\sum_{i=1}^{m} x_i^b \leq \kappa$$

is facet-defining for $P_{MAD}(A,\beta,\kappa)$ if and only if $|\eta(i)| \leq m - \kappa$ holds for every row i (where $\eta(i)$ denotes all nodes adjacent to i in $G(A^T)$).

Proof. We first show that the inequality is facet-defining if the above mentioned condition holds. To this purpose, let $a^T x \leq \alpha$ be a valid inequality that induces a facet such that $\{x \in P_{\text{MAD}}(A, \beta, \kappa) : \sum_{i=1}^m x_i^b = \kappa\} \subseteq \{x \in P_{\text{MAD}}(A, \beta, \kappa) : a^T x = \alpha\}$. We will show that the two inequalities are the same up to a positive scalar multiplicative factor.

Define x by

$$x_i^{b'} = \begin{cases} 1, & \text{if } 1 \leq i \leq \kappa,\ b' = b; \\ 0, & \text{else.} \end{cases}$$

x is a feasible block decomposition that assigns the first κ rows to block b. x satisfies the block capacity constraint with equality and thus $a^T x = \alpha$. Now observe that, for all $1 \leq i \leq \kappa < j \leq m$, the vector $x - e_i^b + e_j^b$ is also a feasible assignment that is tight for the block capacity inequality. It follows that $a_i^b = a_j^b$ for all $1 \leq i, j \leq m$.

Now consider assigning some row j to a block $b' \neq b$. By the assumption $|\eta(j)| \leq m - \kappa$, there is a set $R(j)$ of κ rows not adjacent to j. But then $\sum_{i \in R(j)} e_i^b$ and $\sum_{i \in R(j)} e_i^b + e_j^{b'}$ are both feasible decompositions that satisfy the block capacity constraint with equality and thus $a_j^{b'} = 0$, completing the first part of the proof.

It remains to prove the converse direction. If there is some row j with $|\eta(j)| > m - \kappa$, the inequality $\sum_{i=1}^m x_i^b + \sum_{b' \neq b} x_j^{b'} \leq \kappa$ is valid. But then the block capacity constraint can be obtained by summing up this inequality with $\sum_{b' \neq b} x_j^{b'} \geq 0$, and therefore it cannot be facet-defining. □

6.3.2 Block-Invariant Inequalities

We investigate in this section inequalities for the matrix decomposition polytope that are invariant under block permutation. Consider for each block decomposition x the "aggregated" vector

$$z(x) := \left(\sum_{b=1}^{\beta} x_1^b, \ldots, \sum_{b=1}^{\beta} x_m^b \right) \in \mathbb{R}^m.$$

$z(x)$ only records whether the matrix rows are assigned to some block or not, but no longer to which block. From a polyhedral point of view, the aggregated block decompositions give rise to an "aggregated" version of the block decomposition polytope

$$P_{\text{MAD}}^z(A, \beta, \kappa) := \text{conv}\{z \in \mathbb{R}^m : \text{ there is } x \in P_{\text{MAD}}(A, \beta, \kappa) \text{ with } z = z(x)\}.$$

The aggregated polytope is interesting because any valid inequality $\sum_{i=1}^m a_i z_i \leq \alpha$ for $P_{\text{MAD}}^z(A, \beta, \kappa)$ can be "expanded" into an inequality $\sum_{i=1}^m a_i \sum_{b=1}^{\beta} x_i^b \leq \alpha$ that is valid for $P_{\text{MAD}}(A, \beta, \kappa)$. All inequalities in this subsection are of this type. Obviously, the expansion process yields inequalities that are invariant under block permutations, hence the name. Aggregation techniques such as this are discussed in a more general context and applied to many combinatorial optimization problems in Borndörfer and Weismantel [1997a], [1997b].

From a computational point of view, block-invariant inequalities are promising cutting planes, because the objective of the matrix decomposition problem can be written in terms of aggregated z-variables as $\mathbb{1}^T x = \mathbb{1}^T z(x)$. Thus, a complete description of $P_{\text{MAD}}^z(A, \beta, \kappa)$ would already allow us to determine the correct objective function value of the matrix decomposition problem and z-cuts will help to raise the lower bound of an LP-relaxation.

The aggregated polytope $P_{\text{MAD}}^z(A, \beta, \kappa)$ provides a model of the matrix decomposition problem that rules out degeneracy due to block permutations. While this is a very desirable property of the aggregated z-formulation, its drawback is that it is already $\mathcal{NP}$-complete to decide whether a given vector $z \in \{0,1\}^m$ is an aggregated block decomposition or not. (It can be shown that this is a bin-packing problem.) Our choice to use z-cuts within the x-model tries to circumvent this difficulty and combines the strengths of both formulations. We remark that degeneracy problems of this type arise also in block-indexed formulations of grouping problems in cellular manufacturing, where the difficulty can be resolved by means of alternative formulations, see Crama and Oosten [1996].

We already know one example of an expanded aggregated constraint: Expanding the inequality $z_i \leq 1$ for the aggregated block decomposition polytope yields the block assignment constraint (6.1) (i) $\sum_{b=1}^{\beta} x_i^b \leq 1$ that we have analyzed in the previous subsection. More inequalities are derived from the observation that adjacent rows (with respect to $G(A^T)$) can only be assigned to the same block. A first example of this sort of inequalities are the *z-cover inequalities*.

Theorem 6.3.6 (z-Cover) *Let $G(A^T) = (V, E)$ and let $W \subseteq V$ be a set of rows of cardinality $\kappa + 1$. Then, the z-cover inequality*

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \leq \kappa$$

is valid for $P_{MAD}(A, \beta, \kappa)$ if and only if $(W, E(W))$ is connected. It is facet-defining for $P_{MAD}(A, \beta, \kappa)$ if and only if for each row $i \notin W$ the graph $(W \cup \{i\}, E(W \cup \{i\}))$ has an articulation point different from i.

Proof. The validity part is easy. Since $|W| = \kappa + 1$, not all rows can be assigned to the same block. If some rows of W are assigned to different blocks, there must be at least one row in W that is not assigned because $(W, E(W))$ is connected. Conversely, if W is not connected one easily finds a partition of W into two subsets that can be assigned to different blocks.

The proof that this inequality is facet-defining if and only if for each row $i \notin W$ the graph $(W \cup \{i\}, E(W \cup \{i\}))$ has an articulation point different from i is analogous to the proof of Theorem 6.3.5. The condition guarantees that if row i is assigned to some block, the assignment can be extended in such a way that κ rows from W can be assigned to at least two blocks. On the other hand, if the condition is not satisfied for some $j \notin W$, the inequality $\sum_{i \in W \cup \{j\}} \sum_{b=1}^{\beta} x_i^b \leq \kappa$ is valid, and thus the z-cover inequality cannot be facet-defining. □

In the set covering model, z-cover inequalities correspond to constraints of the form $\sum_{i \in W} \sum_{b=1}^{\beta} y_i^b \geq 1$ that have been used by Nicoloso and Nobili [1992] for their computations. The separation problem is to find a tree of size $\kappa + 1$ of maximum node weight. This problem has been studied by Ehrgott [1992] and was shown to be $\mathcal{NP}$-hard using a reduction to the node-weighted Steiner-tree problem.

The z-cover inequalities are induced by trees, but it is possible to generalize them for subgraphs of higher connectivity.

Theorem 6.3.7 (Generalized z-Cover) *Let $G(A^T) = (V, E)$ and let $W \subseteq V$ be a set of rows of cardinality $\kappa + k$ with $k \geq 1$. Then, the (generalized) z-cover inequality*

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \leq \kappa$$

is valid for $P_{MAD}(A, \beta, \kappa)$ if and only if $(W, E(W))$ is k-node connected. It is facet-defining for $P_{MAD}(A, \beta, \kappa)$ if and only if for each row $i \notin W$ there exists some node cut N in $(W \cup \{i\}, E(W \cup \{i\}))$ of cardinality k with $i \notin N$.

The proof of this generalization follows exactly the lines of the proof of Theorem 6.3.6. In our branch-and-cut algorithm we restrict attention to the cases $k = 1$ and $k = 2$.

Closely related to the z-cover inequality is the *z-clique inequality.* Here, we consider some node set W that is not only k-node connected for some fixed k, but induces a complete subgraph. In this case the condition for being facet-defining slightly changes.

Theorem 6.3.8 (z-Clique) *If Q is a clique in $G(A^T)$, then the z-clique inequality*

$$\sum_{i \in Q} \sum_{b=1}^{\beta} x_i^b \leq \kappa$$

is valid for $P_{MAD}(A, \beta, \kappa)$. It is facet-defining if and only if $|Q| \geq \kappa + 1$ and for each row $i \notin Q$ there exists a set of rows $R(i) \subseteq Q$, $|R(i)| = \kappa$, such that i is not adjacent in $G(A^T)$ to any node in $R(i)$.

Proof. The inequality is clearly valid. To show that it is facet-defining given the mentioned conditions, let $a^T x \leq \alpha$ define a facet such that

$$\{x \in P_{\mathrm{MAD}}(A, \beta, \kappa) : \sum_{i \in Q} \sum_{b=1}^{\beta} x_i^b = \kappa\} \subseteq \{x \in P_{\mathrm{MAD}}(A, \beta, \kappa) : a^T x = \alpha\}.$$

We will show that the two inequalities are the same up to a positive scalar multiplicative factor. To this purpose, consider any κ rows of Q. The block decomposition obtained by assigning these rows to some block b is feasible and tight for the z-clique inequality. Since $|Q| \geq \kappa + 1$, we can use these solutions to show that $a_i^b = a_j^{b'}$ for all $i, j \in Q$ and for all blocks $b, b' \in \{1, \ldots, \beta\}$. Assuming that for each row $i \notin Q$ there exists a set of nodes $R(i) \subseteq Q$, $|R(i)| = \kappa$, that are not adjacent to i, we observe that for all $b' \neq b$, the vectors $\sum_{j \in R(i)} e_j^b$ and $\sum_{j \in R(i)} e_j^b + e_i^{b'}$ are valid block decompositions that satisfy the z-clique inequality with equality. It follows that $a_i^{b'} = 0$ for all $i \notin Q$, for all $b' \neq b$, and even for all blocks b', since b was arbitrary. This completes the first part of the proof.

If, on the other hand, Q has size less than or equal to κ, we obtain from (6.1) (i) that the left-hand side of the inequality is at most $|Q|$. Thus, the inequality is redundant and cannot define a facet. Suppose now the second condition is not satisfied, i. e., there is some $j \notin Q$ such that j is incident to at least $|Q| - \kappa + 1$ nodes in Q. This implies that $Q \cup \{j\}$ is at least $(|Q| - \kappa + 1)$-node connected. Theorem 6.3.7 states that $\sum_{i \in Q \cup \{j\}} \sum_{b=1}^{\beta} x_i^b \leq \kappa$ is valid and this implies that the z-clique inequality is redundant. □

The z-clique separation problem is again a max-clique problem and thus $\mathcal{NP}$-hard. In our implementation we check easily detectable special cases like the following so-called *big-edge inequalities*

$$\sum_{i \in \mathrm{supp}(A_{\cdot j})} \sum_{b=1}^{\beta} x_i^b \leq \kappa,$$

for all blocks b. These inequalities can be separated by inspection.

Another way to generalize the z-cover inequalities is by looking at node induced subgraphs that consist of several components. This idea, that gives rise to the class of *bin-packing inequalities*, came up in our computational experiments. Starting point is again a set of rows W that induces a subgraph of $G(A^T) = (V, E)$. Suppose $(W, E(W))$ consists of l connected components of sizes (in terms of nodes) $a_1, \ldots, a_l$.

We can then associate a *bin-packing problem* with $(W, E(W))$, β, and κ in the following way: There are l items of sizes $a_1, \ldots, a_l$, and β bins of capacity κ each. The problem is to put all the items into the bins such that no bin holds items of a total size that exceeds the capacity κ. If this is not possible, we can derive a valid inequality for $P_{\mathrm{MAD}}(A, \beta, \kappa)$.

Theorem 6.3.9 *Let $G(A^T) = (V, E)$ and $W \subseteq V$ be some subset of rows. If the bin packing problem associated to $(W, E(W))$, β, and κ has no solution, the bin-packing inequality*

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \leq |W| - 1$$

is valid for $P_{MAD}(A, \beta, \kappa)$.

Proof. Consider some block decomposition x. If at least one row in W is not assigned to some block, the inequality is obviously satisfied. Otherwise all rows that belong to the same (connected) component of $(W, E(W))$ must be assigned to the same block. This yields a solution to the bin packing problem associated to $(W, E(W))$, β, and κ, a contradiction. □

We do not know any reasonable conditions that characterize when the bin packing inequalities are facet-defining. Bin-packing separation is $\mathcal{NP}$-hard, see Garey and Johnson [1979].

Next we give another class of *z-cycle inequalities* that generalize the cycle inequalities of the set packing polytope.

Theorem 6.3.10 (z-Cycle) *Let $G(A^T) = (V, E)$ and $C \subseteq V$ be a cycle in $G(A^T)$ of cardinality at least $\kappa + 1$. Then the z-cycle inequality*

$$\sum_{i \in C} \sum_{b=1}^{\beta} x_i^b \leq |C| - \left\lceil \frac{|C|}{\kappa + 1} \right\rceil$$

is valid for $P_{MAD}(A, \beta, \kappa)$.

The z-cycle inequality is valid because at least every $(\kappa + 1)$-st node cannot be assigned to a block. One can also show that the inequality is facet-defining for its support under certain rather restrictive conditions, for example, if C is an odd hole, $|C| \neq 0$ modulo $(\kappa + 1)$, and the right-hand side is less than $\beta\kappa$. z-Cycle separation can be reduced to the traveling salesman problem and is thus $\mathcal{NP}$-hard.

Our next class of inequalities comes up in several instances in our test set.

Theorem 6.3.11 (Composition of Cliques (COQ)) *Let $G(A^T) = (V, E)$ and consider p mutually disjoint cliques $Q_1, \ldots, Q_p \subseteq V$ of size q and q mutually disjoint cliques $P_1, \ldots, P_q \subseteq V$ of size p such that $|P_i \cap Q_j| = 1$ for all i, j. Let $W = \cup_{i=1}^p Q_i$. Then, the following inequality is valid for $P_{MAD}(A, \beta, \kappa)$:*

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \leq \max_{\substack{\{r \in \mathbb{N}^\beta, s \in \mathbb{N}^\beta: \\ \sum r_b = p, \sum s_b = q\}}} \sum_{b=1}^{\beta} \min\{\kappa, r_b s_b\} =: \alpha(p, q, \beta, \kappa). \tag{6.4}$$

Proof. Consider a block decomposition x and let

$$\begin{aligned} \overline{r}_b &:= |\{j : \textstyle\sum_{i \in Q_j} x_i^b \geq 1, j \in \{1, \ldots, p\}\}|, \\ \overline{s}_b &:= |\{j : \textstyle\sum_{i \in P_j} x_i^b \geq 1, j \in \{1, \ldots, q\}\}|, \end{aligned}$$

for $b = 1, \dots, \kappa$. Because Q_j and P_j are all cliques, we have that $\sum_{b=1}^{\beta} \overline{r}_b \le p$ and $\sum_{b=1}^{\beta} \overline{s}_b \le q$. Since $|P_i \cap Q_j| = 1$ for all i, j it follows that $\sum_{i \in W} x_i^b \le \overline{r}_b \overline{s}_b$. Thus,

$$\begin{aligned}
\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b &= \sum_{b=1}^{\beta} \sum_{i \in W} x_i^b \\
&\le \sum_{b=1}^{\beta} \min\{\kappa, \overline{r}_b \overline{s}_b\} \\
&\le \max_{\substack{\{r \in \mathbb{N}^\beta, s \in \mathbb{N}^\beta: \\ \sum r_b \le p, \sum s_b \le q\}}} \sum_{b=1}^{\beta} \min\{\kappa, r_b s_b\} \\
&= \max_{\substack{\{r \in \mathbb{N}^\beta, s \in \mathbb{N}^\beta: \\ \sum r_b = p, \sum s_b = q\}}} \sum_{b=1}^{\beta} \min\{\kappa, r_b s_b\},
\end{aligned}$$

showing the statement. □

The right-hand side of (6.4) is quite complicated, and we do not even know whether it can be computed in polynomial time. For $\beta = 2$ the right-hand side looks more tractable:

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \le \max_{\substack{r=0,\dots,p \\ s=0,\dots,q}} \left(\min\{\kappa, rs\} + \min\{\kappa, (p-r)(q-s)\}\right). \tag{6.5}$$

But we do not know a closed formula in this case either. An interesting special case is $p = 2$. Here the graph $(W, E(W))$ consists of two disjoint cliques that are joint by a perfect matching. Suppose further $q < \kappa < 2q$. Then the right-hand side of (6.5) reads

$$\begin{aligned}
&\max\{0, \max_{s=0,\dots,q} (\min\{\kappa, s\} + \min\{\kappa, q-s\}), \min\{\kappa, 2q\}\} \\
={}& \max\{0, q, \kappa\} = \kappa.
\end{aligned}$$

In this case (6.4) turns out to be even facet-defining if we require in addition that each node $i \notin W$ has at most $2q - \kappa$ neighbors in W, i. e., $|\eta(i) \cap W| \le 2q - \kappa$.

The development of our heuristic separation routine for COQ inequalities resulted in a slight generalization of this class. The support graphs of the left-hand sides of these *extended composition of clique inequalities* are COQs where some nodes have been deleted, the right-hand sides are left unchanged.

Theorem 6.3.12 (Extended Composition of Cliques (xCOQ)) *Consider the graph $G(A^T) = (V, E)$ and p mutually disjoint non-empty cliques $Q_1, \dots, Q_p \subseteq V$ of size at most q and q mutually disjoint non-empty cliques $P_1, \dots, P_q \subseteq V$ of size at most p such that*

(i) $|P_i \cap Q_j| \le 1$ *for all* i, j *and*

(ii) $\sum_{i=1}^{q} \sum_{j=1}^{p} |P_i \cap Q_j| = \sum_{i=1}^{q} |P_i| = \sum_{j=1}^{p} |Q_j|$,

i. e., every element in one of the sets P_i appears in exactly one of the sets Q_j and vice versa. Let $W = \cup_{i=1}^{p} Q_i$. Then, the following inequality is valid for $P_{MAD}(A, \beta, \kappa)$:

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \le \alpha(p, q, \beta, \kappa). \tag{6.6}$$

Proof. The proof works by turning $P_1, \ldots, P_q$ and $Q_1, \ldots, Q_p$ into a proper COQ by adding some nodes that correspond to "artificial rows" and projecting the resulting inequality down to the original space of variables.

Let

$$\delta := \sum_{i=1}^{q} \sum_{j=1}^{p} (1 - |P_i \cap Q_j|)$$

be the number of nodes that "miss" to turn $P_1, \ldots, P_q$ and $Q_1, \ldots, Q_p$ into a COQ and add a row $\mathbb{1}^T$ of all ones to A for each of them to obtain a matrix $\bar{A}$ such that

$$\bar{A}_{i\cdot} = A_{i\cdot}, \quad i = 1, \ldots, m \qquad \text{and} \qquad \bar{A}_{i\cdot} = \mathbb{1}^T, \quad i = m+1, \ldots, m+\delta.$$

Consider the matrix decomposition problem $(\bar{A}, \beta, \kappa)$. Its row intersection graph $G(\bar{A}^T)$ contains $G(A^T)$ as a subgraph and the additional artificial nodes in $G(\bar{A}^T)$ are incident to every node of $G(\bar{A}^T)$ that corresponds to a row that is not all zero (except itself).

The sets $P_1, \ldots, P_q$ and $Q_1, \ldots, Q_p$ are again cliques in $G(\bar{A}^T)$. Associating each of the artificial nodes $i = m+1, \ldots, m+\delta$ to a different index pair ij such that $|P_i \cap Q_j| = 0$ and adding this node to both P_i and Q_j, we can extend $P_1, \ldots, P_q$ and $Q_1, \ldots, Q_p$ to a COQ $\overline{P}_1, \ldots, \overline{P}_q$ and $\overline{Q}_1, \ldots, \overline{Q}_p$ in $G(\bar{A}^T)$ with $\overline{W} := \cup_{j=1}^{p} \overline{Q}_j = W \cup \{m+1, \ldots, m+\delta\}$. Then, the COQ inequality

$$\sum_{i \in \overline{W}} \sum_{b=1}^{\beta} x_i^b \leq \alpha(p, q, \beta, \kappa) \tag{6.7}$$

is valid for $P(\bar{A}, \beta, \kappa)$ and, of course, also for

$$P(\bar{A}, \beta, \kappa) \cap \{x_i^b = 0 : i = m+1, \ldots, m+\delta, \ b = 1, \ldots, \beta\}.$$

Since the artificial variables in this polytope attain only values of zero, this remains true if one sets their coefficients in (6.7) also to zero. But as this results in the desired extended COQ inequality (6.6) and

$$P(\bar{A}, \beta, \kappa) \cap \{x_i^b = 0 : i = m+1, \ldots, m+\delta, \ b = 1, \ldots, \beta\} = P_{\text{MAD}}(A, \beta, \kappa) \times \{0\}^{\delta \times \beta},$$

the theorem follows by a projection on the space of the original variables. □

The last *star inequality* that we present in this section is special in the sense that it is the only one with non-0/1 coefficients. It was designed to deal with rows with many neighbors.

Theorem 6.3.13 (Star) *Let $G(A^T) = (V, E)$ and consider some row $i \in V$ with $|\eta(i)| > \kappa$. Then the star inequality*

$$(|\eta(i)| - \kappa + 1) \sum_{b=1}^{\beta} x_i^b + \sum_{j \in \eta(i)} \sum_{b=1}^{\beta} x_j^b \leq |\eta(i)|$$

is valid for $P_{MAD}(A, \beta, \kappa)$.

Proof. If i is assigned to some block b, then all rows in $\eta(i)$ can only be assigned to b, but at most $\kappa - 1$ of them. The case where i is not assigned is trivial. □

The star inequality can be viewed as a lifting (for lifting see Appendix C) of the (redundant) inequality

$$\sum_{j \in \eta(i)} \sum_{b=1}^{\beta} x_j^{\beta} \leq |\eta(i)|$$

and we want to close this section with another simple lifting theorem for block-invariant inequalities with 0/1 coefficients.

Theorem 6.3.14 (Strengthening) *Let $G(A^T) = (V, E)$, W be a subset of V, and $\sum_{i\in W}\sum_{b=1}^{\beta} x_i^b \le \alpha$ be a valid inequality for $P_{MAD}(A, \beta, \kappa)$. If for some row $j \notin W$ the condition*

$$|W \setminus \eta(j)| + \kappa \le \alpha$$

holds, then $\sum_{i\in W\cup\{j\}}\sum_{b=1}^{\beta} x_i^b \le \alpha$ is also valid for $P_{MAD}(A, \beta, \kappa)$.

Proof. If j is assigned to some block b, the rows in $\{j\} \cup \eta(j)$ can only be assigned to b, but at most κ of them. □

6.4 Algorithmic Aspects

The polyhedral investigations of the last section form the basis for the implementation of a branch-and-cut algorithm for the solution of the matrix decomposition problem. This section describes the four main ingredients of this code: Separation and LP-management, heuristics, problem reduction, and some issues on the search-tree management.

6.4.1 Separation and LP-Management

We use all of the inequalities described in Section 6.3 as cutting planes in our algorithm. It will turn out that some of them appear in large numbers. We thus opted for the following separation strategy: We try to identify many inequalities using fast heuristics, but add only selected ones to the LP. More expensive separation methods are used depending on their success.

We classify our separation routines according to their basic algorithmic principles: Inspection (enumeration), greedy heuristics, other heuristics, exact polynomial methods, and "hybrid" methods (combinations of exact and heuristic methods).

The most simple methods are used for *big-edge, two-partition*, and *star inequalities*: These classes can be separated by simple inspection, the details for two-partition inequalities were already described in Section 6.3.

Clique, z-clique, and *z-cover inequalities* are separated using greedy heuristics. In the last two cases, these methods start by sorting the rows of A with respect to increasing z-value, i.e., such that $z(x)_{i_1} \ge z(x)_{i_2} \ge \cdots \ge z(x)_{i_m}$. Then the greedy heuristic is called m times, once for each row i_j. In each call, i_j is used to initialize a tree/clique with respect to $G(A^T)$, that is iteratively extended greedily in the order of the z-sorting of the rows until z_{i_j} becomes zero and the growing procedure stops. There is also a second variant for z-cliques that is an adaption of a similar routine by Hoffman and Padberg [1993]. Here we call the greedy heuristic once for each column of A and initialize the clique with the support of this column. Having detected a violated clique inequality in one of these ways, we lift randomly determined additional rows with zero z-value sequentially into the inequality. This is done by applying the strengthening procedure of Theorem 6.3.14 which in this case amounts to a further growth of the clique by randomly determined rows of zero z-value. We tried to strengthen cover inequalities, but the computational effort was not justified by the one or two coefficients that were usually lifted. But, as was already mentioned in Section 6.3, we (heuristically) keep track of the connectivity of the growing graph. If the connectivity is 2 after the graph reached size $\kappa + 1$, we add another two-connected node if possible. Separation of clique inequalities is done using exactly the same routine as for z-cliques, but applied to $G_c(A, \beta)$ with node weights given by the x-variables.

z-Cycle inequalities are separated in the following heuristic way. We look at some path P with endnodes u and v, where initially u and v coincide. In each iteration we extend the path at one of its endnodes by a neighbor w with maximal

$z(x)_w$-value. Let j_w be a column of A that connects w to the path P. Since j_w forms a clique in $G(A^T)$ there are additional nodes that can be potentially added to the path if the support of j_w is greater than two, i. e., $|\operatorname{supp}(A_{\cdot j_w})| > 2$. We store these additional nodes in a buffer which will be exploited later in the heuristic. Now we test whether the new path P extended by w can be closed to a cycle C that satisfies $|C| > \kappa$ and $|C| \neq 0$ modulo $(\kappa+1)$. This is done by looking for a column j of A that contains both endnodes of P (one of them w). $\operatorname{supp}(A_{\cdot j})$ again forms a clique, and the additional nodes in this clique together with the nodes in the buffer give the flexibility to add further nodes to the cycle. This freedom is exploited in our routine. We try the procedure for several starting nodes $u = v$, whose number depends on the success of the heuristic.

Separation of the *composition of clique inequalities* is not easy: We do not even know a way to compute the right-hand side $\alpha(p, q, \beta, \kappa)$ in polynomial time! But there are problems in our test set, where compositions of cliques occur and there seems to be no way to solve this (small!) problem without them. Our heuristic was developed to capture these cases. It lead to the development of the more general class of extended COQ inequalities, which are easier to find. The idea is as follows.

Let us start with a composition of cliques $Q_1, \ldots, Q_p$ and $P_1, \ldots, P_q$ as stated in Theorem 6.3.11. Assume that these cliques are contained in the columns $1, \ldots, p, p+1, \ldots, p+q$ of the matrix A, i. e., $\operatorname{supp}(A_{\cdot i}) \supseteq Q_i$, $i = 1, \ldots, p$, and $\operatorname{supp}(A_{\cdot i}) \supseteq P_i$, $i = p+1, \ldots, p+q$. Consider a *column/column-incidence matrix* S of A defined by

$$s_{ij} = \begin{cases} k, & \text{for } k \in \{l : a_{li} \neq 0 \neq a_{lj}\} \text{ arbitrary, but fixed;} \\ 0, & \text{if } A_{\cdot i}^T A_{\cdot j} = 0, \end{cases}$$

i. e., $s_{ij} = k \neq 0$ if and only if columns i and j intersect in some row k and in case there is no unique k we pick an arbitrary, but fixed one. Suppose for the moment that all entries in the submatrix $S_{\{1,\ldots,p\}\times\{p+1,\ldots,p+q\}}$ of S are mutually different, that is, there is no row index k that appears more than once. Then the composition of cliques corresponds to the rectangle submatrix $S_{\{1,\ldots,p\}\times\{p+1,\ldots,p+q\}}$ of S that is completely filled with non-zeros: The rows that appear on the left-hand side of the COQ inequality (6.4) are exactly those appearing in the matrix $S_{\{1,\ldots,p\}\times\{p+1,\ldots,p+q\}}$. In other words, the node set W in (6.4) is $W = \{s_{ij} : i = 1, \ldots, p, \, j = p+1, \ldots, p+q\}$. Thus, given some vector $x \in \mathbb{R}^{m\times\beta}$, the left-hand side of (6.4) is $\sum_{i=1}^{p} \sum_{j=p+1}^{p+q} \sum_{b=1}^{\beta} x_{s_{ij}}^b$ and a final calculation of the right-hand side allows to check for a possible violation of the inequality.

Our heuristic tries to go the reverse direction: It identifies large filled rectangles in S and derives COQ and xCOQ inequalities from them. There are three difficulties. First, a clique in a composition cannot only be a subset of a column of A, but any clique in $G(A^T)$. However, we have not incorporated this generality in our heuristic, because we do not know how to select a promising set of cliques in $G(A^T)$. Second, columns in A that form a composition of cliques may not appear in the right order: The rectangle identifies itself only after a suitable permutation of S. In this case, we have to reorder the columns and rows of S. We obtain a filled rectangle submatrix $S_{I\times J}$ of S by starting with each of column j of S once, extend this $1 \times |\operatorname{supp}(A_{\cdot j})|$ rectangle submatrix by columns that fit best in a greedy way, sort its rows lexicographically, and consider all maximal filled submatrices in the upper left corner as potential COQ-rectangles. A third serious problem arises when two columns of A intersect in more than one row. In this case the entries of the matrix S are no longer uniquely determined and it can happen that the entries of the rectangular submatrix $S_{I\times J}$ under consideration are no longer mutually different. Then $S_{I\times J}$ corresponds no longer to a composition of cliques and the inequality $\sum_{ij\in I\times J} \sum_{b=1}^{\beta} x_{s_{ij}}^b \leq \alpha\big(|I|, |J|, \beta, \kappa\big)$ is in general not valid. But one can set duplicate entries in S to zero until, for every row k, there is only one representative

$s_{ij} = k$ left; denote the resulting matrix by S'. Then the sets

$$\begin{aligned} \overline{Q}_i &:= \{s'_{ij} : s'_{ij} \neq 0,\ j \in J\}, \quad i \in I \qquad \text{and} \\ \overline{P}_j &:= \{s'_{ij} : s'_{ij} \neq 0,\ i \in I\}, \quad j \in J \end{aligned}$$

of non-zero entries in the rows and columns of S' form an extended composition of cliques and the corresponding xCOQ inequality

$$\sum_{k \in \operatorname{im}(S_{I \times J})} \sum_{b=1}^{\beta} x_k^b \leq \alpha(|I|, |J|, \beta, \kappa)$$

is valid for $P_{\mathrm{MAD}}(A, \beta, \kappa)$, where $\operatorname{im}(S_{I \times J}) = \{s_{ij} : ij \in I \times J\}$ denotes the set of row indices that appear in the submatrix $S_{I \times J}$. The interesting feature of separating extended COQ inequalities instead of COQs is that the generalization gives us the algorithmic freedom to handle multiple occurrences of rows in filled rectangles of S and this is the key to a successful heuristic separation of an otherwise rigid structure. The price for this, of course, is a reduced support in the left-hand side. To pay this price only when necessary, we heuristically determine a column/column-intersection matrix S with a large variety of rows in $\operatorname{im}(S)$. The right-hand side itself is computed in amortized (pseudo-polynomial) time of $O(\beta \kappa n^2)$ steps by a dynamic program (for our tests $\beta \leq 4$ and $\kappa = O(n)$, and thus this effectively amounts to $O(n^3)$).

The reader might have noticed that several degrees of freedom in this separation routine can be used to search for rectangles with large z-value and this is what we would like to find. However, the running time of the method is too large to apply it after each LP and when we did, we did not find additional cuts. We thus call the routine only once, determine some promising COQs by combinatorial criteria, store them in memory, and separate them by inspection.

To separate *clique inequalities* (for $\beta > 2$), we use an exact branch-and-bound algorithm for the maximum weight clique problem. Although in principle exponential, this algorithm works fast for the separation problems coming up in our matrix decomposition instances because the maximum clique size is bounded by β. We have also tried to separate z-cliques exactly, but we never observed that additional cuts were found: In the small examples, the greedy heuristic is good enough, while in the larger ones with high capacities cliques of size κ do not seem to exist. Another exact, but this time polynomial, algorithm is used to separate *cycle inequalities*: We apply the odd-cycle algorithm described in Lemma 9.1.11 in Grötschel, Lovász, and Schrijver [1988].

Finally, a mixture of exact and heuristic ideas is used in a hybrid algorithm to separate the *bin-packing inequalities*. We start by determining a node set W that can result in a violated inequality. A necessary condition for this is

$$\sum_{i \in W} z(x)_i > |W| - 1 \iff 1 > \sum_{i \in W} (1 - z(x)_i)$$

and it is reasonable to construct W by iteratively adding rows that have a z-value close to one. We thus sort the nodes with respect to increasing z-value and add them to W in this order as long as the condition stated above is satisfied. This node set W induces a subgraph $(W, E(W))$ of $G(A^T)$ and we determine the components of this subgraph. The resulting bin-packing problem (see page 116) is solved using an exact dynamic programming algorithm (with a time bound).

In addition to these classical types of cutting planes we also use a number of *"tie-breaking" inequalities* to cut off decompositions that are identical up to block permutations or give rise to multiple optima for other reasons as a means to counter dual degeneracy and stalling. These inequalities are in general not valid for $P_{\mathrm{MAD}}(A, \beta, \kappa)$, but for at least one optimal solution. The most simple kind of these cuts are the *permutation inequalities*

$$\sum_{i=1}^{m} x_i^b \leq \sum_{i=1}^{m} x_i^{b+1}, \quad b = 1, \ldots, \beta - 1,$$

stating that blocks with higher indices are of larger size. To break further ties, we supplement them with inequalities stipulating that in case of equal sized blocks the row with the smallest index will be assigned to the block with smaller index. These *strengthened permutation inequalities* read

$$x_k^{b+1} + \sum_{i=1}^{m} x_i^b - \sum_{i=1}^{m} x_i^{b+1} \leq \sum_{i=1}^{k-1} x_i^b, \quad b = 1, \ldots, \beta - 1, \ k = 2, \ldots, m - 1.$$

If $\sum_{i=1}^{m} x_i^b - \sum_{i=1}^{m} x_i^{b+1} < 0$, the inequality is redundant, but in case of equality, the row with the smallest index in blocks b and $b+1$ must be in block b. The case $k = m$ is left out because it yields a redundant inequality. Both permutation and strengthened permutation inequalities can be separated by inspection.

Another idea that we use to eliminate multiple optima is based on the concept of *row preference*. We say that row i is *preferred* to row j or, in symbols, $i \prec j$ if

$$\eta(i) \subseteq \eta(j)$$

with respect to the row intersection graph $G(A^T)$. We may in this situation not know whether or not row i or j can be assigned to a block in some optimal solution, but we can say that for any decomposition x with $z(x)_j = 1$, say $x_j^b = 1$, either $z(x)_i = 1$ or we can get a feasible decomposition $x' = x - e_j^b + e_i^b$ with the same number of rows assigned. In this sense, row i is more attractive than row j. If we break ties on row preference by indices (i. e., $i \prec j \iff \eta(i) \subsetneq \eta(j) \vee (\eta(i) = \eta(j) \wedge i < j)$), row preferences induce a partial order that we represent in a transitive and acyclic digraph

$$D(A) := (V, \{(i,j) : i \prec j\}).$$

Since the number of row preferences tends to be quadratic in the number of rows, we thin out this digraph by removing all transitive (or implied) preferences. The remaining row preferences are forced in our code by adding the *row preference inequalities*

$$\sum_{b=1}^{\beta} x_i^b \geq \sum_{b=1}^{\beta} x_j^b \quad \text{for } (i,j) \text{ with } i \prec j.$$

These can sometimes be strengthened to

$$x_i^b \geq x_j^b \quad \text{for all } b = 1, \ldots, \beta,$$

if we can be sure that rows i and j cannot be assigned to different blocks in any decomposition. This will be the case, for example, if i and j are adjacent in $G(A^T) = (V, E)$ or if both i and j are adjacent to some third row k preferable to both of them (i. e., $i \prec j$, $k \prec i$, $k \prec j$, $ik \in E$ and $jk \in E$). Once $D(A)$ is set up, row preference inequalities can be separated by inspection.

Our last separation routine uses a cut pool that stores all inequalities found by the hitherto explained algorithms: The pool separation routine just checks all inequalities in the pool for possible violation.

The separation algorithms described in the previous paragraphs turned out to be very successful: Not only block-discernible (permutable) inequalities like two-partitions are found in large numbers, also block-invariant cuts like z-covers occur in abundance. Controlling the growth of the LP-relaxation is thus the main goal of our separation and LP-maintenance strategy. We start with a minimal LP-relaxation containing (besides the bounds) only the block assignment and block capacity constraints plus the $\beta - 1$ permutation inequalities. The cuts that are separated by inspection, i. e., big-edge inequalities, star inequalities, tie-breaking inequalities, and composition of clique inequalities are placed in the cut pool; they will be found by pool separation. The separation algorithms are called dynamically throughout the course of the branch-and-cut algorithm. After an LP is solved, we call the pool separation routine, followed by two-partition inequality separation and a couple of heuristics: The z-cover heuristic is called as it is, but application of the more expensive z-clique and z-cycle algorithms is controlled by a simple time- and success-evaluation. This control mechanism is motivated by the observation that our test set fell into two groups of examples, where one of these routines was either indispensable or essentially did not find a single cut. We empirically try to adapt to these situations by calling the separation routines only if their past success is proportional to the running time, or more precisely, if after the first call

$$\frac{\#\text{ of successful calls} + 1}{\#\text{ of calls}} > \frac{\text{time spent in routine}}{\text{total time spent in separation}}.$$

A call is counted as successful if a violated cut is found. If $\beta > 2$, there can be clique inequalities that are not two-partition constraints and in this case we next call the exact clique separation routine, that returns at most one cut. The branch-and-bound algorithm used there turned out to be fast enough to be called without any further considerations. Finally, we separate bin-packing inequalities. To avoid excessive running times due to the dynamic program, the routine is called with a time limit: The dynamic program will be stopped if the time spent in bin-packing separation exceeds the cumulated separation time of all other separation routines.

All violated cuts determined in this separation process are not added directly to the LP-relaxation, but stored in a cut buffer first. This buffer is saved to the pool, and then a couple of promising cuts are selected to strengthen the LP-relaxation. Our criteria here have an eye on the amount of violation and on the variety of the cuts. Since inequalities of one particular type tend to have similar support, we restrict the number of cuts per type and prefer to add inequalities of other types, even if they are not among the most violated. To accomplish this we add the

$$\frac{\#\text{ of cuts in cut buffer}}{\#\text{ number of types of cuts}}$$

most violated cuts of each type to the LP-relaxation. We also delete cuts from the LP-relaxation if they become non-binding by a slack of at least 10^{-3}, but keep them in the cut pool for a possible later pool separation.

Another feature of our code that aims for small LPs is to locally setup the LP-relaxation prior to computation at any node of the search-tree. This means that when branching on some node v we store at each of its sons a description of the last LP solved at v and of the optimal basis obtained. When we start to process v's sons, we set up this LP from scratch and load the associated (dual feasible) basis. In this way, we continue the computation exactly at the point where it stopped and the LP will be the result of a contiguous process independent of the node selection strategy. We have compared this approach to one where the start-LP at each newly selected node is just the last LP in memory and this leads to larger LPs and larger running times.

While these strategies were sufficient to keep the size of the LP-relaxation under control, explosive growth of the cut pool was a serious problem in our computations until we implemented the following cut pool management. We distinguish between disposable and indisposable cuts in the pool. *Indisposable cuts* are inequalities that are needed to set up the LP-relaxation at some node in the search-tree yet to be processed and all big-edge, star, and tie-breaking inequalities. All other cuts are *disposable* and can potentially be deleted from the cut pool, possibly having to be recomputed later. In order to control the pool size we restrict the number of disposable cuts in the pool by eliminating cuts that have not been in any LP for a certain number of iterations. This number depends on the size of the pool and the ratio of disposable to indisposable cuts.

6.4.2 Primal Heuristics

We have implemented several primal heuristics for our matrix decomposition code. Since different nodes in a branch-and-bound tree correspond to different fixings of variables to zero or one, the heuristics should respect these fixings to increase the probability of finding different solutions. Applied at the root node where (at least initially) no variables are fixed, our methods can be seen as LP-based or pure combinatorial heuristics for the matrix decomposition problem.

Our heuristics fall into three groups: "Primal" methods that iteratively fix block assignments, "dual" methods that iteratively exclude assignments until decisions become mandatory due to lack of alternatives, and an improvement method that is applied as an "after-burner" to enhance the quality of the two groups of opening heuristics.

The primal methods consist of a greedy algorithm and a bin-packing heuristic, both are LP-based. The greedy algorithm starts by ordering the x_i^b variables; with probability $\frac{1}{2}$ a random ordering is chosen, otherwise a sorting according to increasing x-value is used. The rows are assigned greedily to the blocks in this order. This heuristic is similar in spirit to the popular "LP-plunging" method, i.e., the iterative rounding of some fractional LP-value to an integer followed by an LP-reoptimization, but much faster. We have also tried LP-plunging, but for the matrix decomposition problem the results were not better than with the simple greedy method, while the running time was much larger. The bin-packing heuristic starts by determining a set of nodes W that will be assigned to the blocks and used to set up a corresponding bin-packing problem. In order to find a better decomposition than the currently best known with, say, $z^\star$ rows assigned, W should be of cardinality at least $z^\star + 1$ and therefore we take the $z^\star + 1$ rows with the largest $z(x)$-values to be the members of W. The corresponding bin-packing problem is set up and solved with the same dynamic program that we used for the separation of the bin-packing inequalities; it is also called with a time limit, namely 10 times as much as all other primal heuristics (that are very fast) together. Clearly, we also watch out for better solutions that might be detected in bin-packing separation.

The dual methods also respect variable fixings, but are not LP-based. The idea behind them is not to assign rows to blocks, but to iteratively eliminate assignments of "bad" rows. Suppose that a decision was made to assign certain rows (assigned rows) to certain blocks, to exclude other rows from assignment (unassigned rows), while for the remaining rows a decision has yet to be made (free rows). Removing the unassigned nodes from the row intersection graph $G(A^T)$ leaves us with a number of connected components, some of them larger than the maximum block capacity κ, some smaller. Both variants of the dual method will break up the components that are larger than the block capacity κ by unassigning free rows until no more such components exist. At this point, a simple first-fit decreasing heuristic is called to solve the corresponding bin-packing problem. The two variants differ in the choice of the next bad row to remove. Variant I chooses the free row in some component of size larger than κ with the largest degree with respect to the row intersection graph $G(A^T)$, variant II excludes assignment of a free row of some component with size larger than κ with the largest indegree with respect to $D(A)$, or, in other words, the least preferable row. We have also tried to use a dynamic program to solve the bin-packing problems, but it did not provide better results in our tests.

Our improvement heuristic is a variation of a local search technique presented by Fiduccia and Mattheyses [1982]. Given some block decomposition, it performs a sequence of local exchange steps each of the following type. Some assigned row is chosen to be made unassigned opening up possibilities to assign its unassigned neighbors. These assignments are checked and feasible assignments are executed. The details are as follows. The heuristic performs a number of passes (10 in our implementation). At the beginning of each pass, all rows are eligible for unassignment in the basic exchange step. Each row may be selected only once for unassignment in each pass and will then be "locked". Candidates for becoming unassigned are all currently assigned and unlocked rows. These candidates are rated according to the number of possible new assignments (computed heuristically) and we choose the one that is best with respect to this rating. As a special annealing-like feature, the algorithm will also perform the exchange step if it leads to a change of the current solution to the worse. If no exchange step is possible because all assigned rows are already locked, the pass ends and the next pass is started.

The strategy to call the heuristics is as follows. The primal methods are called after each individual LP, whereas the dual heuristics are called only once at each node in the branch-and-bound tree, because they behave in a different way only due to changes in the variable fixings.

6.4.3 Problem Reduction

We use a couple of problem reduction techniques to eliminate redundant data. First we apply an initial preprocessing to the matrix A before the branch-and-cut algorithm is initiated. The purpose of this preprocessing step is to eliminate columns from the matrix A without changing the row intersection graph. We first perform a couple of straightforward tests to identify columns that are contained in other columns and can thus be deleted: We remove empty columns, then unit columns, duplicate columns, and finally by enumeration columns that are contained in others.

These simple initial preprocessing steps are amazingly effective as we will see in the section on computational results. In principle, the number of rows can be reduced also. For example, empty rows could be eliminated and later used to fill excess capacity in any block, duplicate rows or rows with just one non-zero entry could be eliminated by increasing the capacity requirements of one of its adjacent rows. These reductions, however, lead to changes in the IP model and affect all separation routines discussed so far so that we refrained from implementing them.

In addition to this initial preprocessing we do local fixings at the individual nodes of the branch-and-bound search-tree after each call to the LP-solver. Apart from reduced cost fixing (see Appendix B) and fixing by logical implication (i. e., if x_i^b is fixed to one, $x_i^{b'}$ will be fixed to zero for all blocks $b' \neq b$), we try to identify rows that cannot be assigned to any block given the current state of fixings. To this purpose we look at all rows W that are currently fixed for assignment to some block. We then check for each unassigned row i whether the subgraph $(W \cup \{i\}, E(W \cup \{i\}))$ of $G(A^T) = (V, E)$ contains a component with more than κ rows. If so, row i can be fixed to be unassigned.

6.4.4 Further Issues

Despite our efforts to understand the polyhedral combinatorics of the matrix decomposition problem, we do not have a strong grip on the corresponding polytope and after an initial phase of rapid growth of the lower bound, stalling occurs in the presence of significant duality gaps. We believe that – up to a certain point – it is favorable in this situation to resort to branching early, even if there is still slow progress in the cutting plane loop. In fact, we apply a rather "aggressive" branching strategy, splitting the currently processed node if the duality gap could not be reduced by at least 10% in any four consecutive LPs. On the other hand, we "pause" a node (put it back into the list of nodes yet to be processed) if the local lower bound exceeds the global lower bound by at least 10%.

Branching itself is guided by the fractional LP-values. We first look for a most fractional $z(x)$-value. If, e.g., $z(x)_i$ is closest to 0.5 (breaking ties arbitrarily), we create $\beta + 1$ new nodes corresponding to the variable fixings

$$x_i^1 = 1, x_i^2 = 1, \ldots, x_i^\beta = 1, \quad \text{and} \quad \sum_{b=1}^{\beta} x_i^b = 0.$$

In other words, we branch on the block assignment constraint corresponding to row i. The advantage of this scheme is that it leads to only $\beta+1$ new nodes instead of 2^β-nodes in an equivalent binary search-tree. If all $z(x)$-values are integral, we identify a row with a most fractional x-variable and perform the same branching step. We have also tried other branching rules by taking, for instance, the degree of a row in $G(A^T)$ into account, but the performance was inferior to the current scheme.

6.5 Computational Results

In this section we report on computational experiences with our branch-and-cut algorithm for the solution of matrix decomposition problems arising form mixed integer programming matrices. Our aim is to find answers to two complexes of questions. First, we would like to evaluate our branch-and-cut approach: What are the limits in terms of the size of the matrices that we can solve with our algorithm? What is the quality of the cuts, do they provide a reasonable solution guarantee? Second, we want to discuss our concept of decomposition into bordered block diagonal form: Do the test instances have this structure or are most integer programming matrices not decomposable in this way? And do our heuristics provide reasonable decompositions that could be used within an integer programming code?

As test set we use the integer programming matrices from the `Miplib` presolved by `SIP`. In addition, we report on some small matrices arising from Steiner tree packing problems, see Chapter 3. As we know the Steiner tree packing problems are known to be in bordered block diagonal form and we wanted to see whether our code is able to discover this structure.

The test runs were performed on a Sun Enterprise 3000 on one 168 MHz UltraSPARC processor and we used a time limit of 3600 CPU seconds. The format of the upcoming tables is as follows: For each test example, the data is split into two tables. Part I gives information about the problem sizes and the cutting plane phase, Part II reports on the success of the primal heuristics and gives timings. In detail, Column 1 provides the name of the problem, Columns 2 to 4 contain the number of rows, columns and non-zeros of the matrix after presolve (for the original sizes, see Table D.4 in Appendix D and Table 5.3). The succeeding five columns give statistics about the number of cuts generated by our code. There are, from left to right, the number of initial cuts (*Init*) including block assignment, block capacity, big-edge, star, and tie-breaking inequalities (but not composition of clique inequalities, although they are also separated from the pool), the number of z-cover (*Cov*), the number of two-partition (*2part*), the sum of the number of bin-packing, cycle, z-cycle, clique, z-clique, and composition of clique inequalities (*BCC*), and finally the number of violated inequalities separated from the pool (*pool*). The following two columns (Columns 10 and 11) show the number of branch-and-bound nodes (*Nod*) and the number of LPs (*Iter*) solved by the algorithm. Part II of the tables starts again with the name of the problem. The next eight columns give solution values. We do not report the number of assigned rows, but the number of rows in the border, because it is easier to see whether the matrix could be decomposed into block diagonal form (in this case the value is zero) or close to this form (then the value is a small positive integer). *Lb* gives the global lower bound provided by the algorithm. It coincides with the value of the upper bound *Ub* (next column) when the problem is solved to proven optimality. Skipping two columns for a moment, the next four columns refer to the heuristics. *G*, *D1*, *D2* and *B* stand for the greedy, the dual (variant I and II), and the bin-packing heuristic. The corresponding columns show the best solutions obtained by these heuristics throughout the computations at the root node. If this value coincides with the number of rows of the matrix, all rows are in the border and the heuristic failed. The two (skipped) columns right after *Ub* show which heuristic *He* found the best solution after *No* many branch-and-bound nodes (1 means it was found in the root node). The additional letter *I* indicates that the value was obtained by a succeeding call to the improvement heuristic. An asterisk $*$ means that the LP solution provided an optimal block decomposition. The remaining four columns show timings. The last of these columns *Tot* gives the total running time measured in CPU seconds. The first three columns show the percentage of the total time spent to solve the linear programs (*LP*), the time of the separation algorithms (*Sep*), and the time for the heuristics (*Heu*).

6.5.1 The `Miplib` Problems

In this test series we examine whether matrices arising from mixed integer programs can be decomposed into (bordered) block diagonal form.

As mentioned in the introduction of this chapter, decomposing the original constraint matrix A of some general integer program can be useful to tighten its LP-relaxations within a branch-and-cut algorithm. The structure of the decomposed matrix is that of a multiple knapsack or general assignment problem, and inequalities known for the associated polytopes, see Chapter 2, are valid for the mixed integer program under consideration. The first interesting case in this context are two blocks and we set $\beta := 2$. We used $\kappa := \frac{(\#rows)\cdot 1.05}{2}$ rounded up as block capacity, which allows a deviation of 10% of the actual block sizes in the decomposition.

Table 6.1 and 6.2 show our results that we obtained for matrices of mixed integer programs taken from the `Miplib` and preprocessed with `SIP` as described in Section 5.1.

The problems up to 100 rows are with a few exceptions easy. The range of 100 to 300 rows is where the limits of our code become visible and this is the most interesting "hot area" of our tables: The problems here are already difficult, but because of the combinatorial structure and not because of sheer size. The results for the problems with more than 300 rows are of limited significance, because these matrix decomposition problems are large-scale and the algorithm solves only a few LPs within the given time limit. Though we can only solve a couple of readily decomposable large instances to optimality, it is worth noticing that a considerable number of instances indeed decomposes. The difficulty of matrix decomposition problems depends as much on the structure of the matrix as on the number of rows,

Example	Presolved				Cutting Planes				B&B	
	Rows	Col	NZs	Init	Cov	2part	BCC	Pool	Nod	Iter
mod008	6	1	6	23	0	0	0	0	1	1
p0033	14	11	35	37	23	5	0	1	1	3
flugpl	15	5	19	43	15	0	1	0	1	2
enigma	21	99	287	40	916	223	0	280	28	85
rgn	24	33	102	64	722	222	1	445	13	47
gt2	28	173	346	39	1417	364	0	262	28	89
lseu	27	50	163	64	73	37	0	9	1	4
nw04	36	102	915	164	0	0	61	0	1	2
egout	40	24	76	139	39	2	0	0	1	2
pk1	45	32	105	106	35312	22358	16	43046	18787	21746
bell5	85	73	209	165	2367	447	1	643	10	38
misc03	95	133	1801	216	27044	17029	0	39495	4129	5154
bell3a	97	81	235	187	2409	520	1	875	10	31
l152lav	97	695	3712	308	30951	22625	0	70669	1423	2223
harp2	100	1225	2450	116	250	158	0	3	1	4
khb05250	100	1275	2574	196	12590	2028	1	4801	73	224
p0201	113	177	1527	200	8936	2280	1	4866	46	128
stein27	118	27	378	353	423737	163187	3	744679	4198	7753
air03	122	666	5973	830	20599	9360	0	24631	142	390
vpm1	129	98	280	465	1288	43	1	285	4	13
vpm2	129	98	280	465	1288	43	1	285	4	13
pp08a	136	120	304	392	17114	1860	1	5905	46	179
mod010	146	1973	8404	453	71932	59140	1	205880	2029	2947
blend2	169	88	1039	515	0	0	0	0	4	4
noswot	171	50	440	702	106611	43732	1	932484	1603	2736
10teams	210	1600	9600	210	120998	46760	1	245717	421	979
misc07	223	229	8245	506	24061	17172	0	41054	199	235
pp08aCUTS	246	230	828	612	14372	1714	1	3185	19	78
p0548	257	250	1009	523	197234	28114	1	324677	544	1055
dcmulti	272	514	1242	536	249229	18524	1	165055	424	1111
modglob	287	354	892	991	25794	847	1	7372	40	122
fiber	290	416	1250	579	104084	17088	1	75280	142	419
rout	290	540	2085	1580	86879	231	1	17877	823	1027
p0282	305	168	841	609	122304	33242	1	140550	244	595
stein45	331	45	1034	992	46894	22680	0	77903	52	164
qnet1_o	332	800	2400	620	72865	19027	1	77501	88	258
qnet1	370	924	2806	408	54982	20045	1	50629	52	172
air05	408	3104	23458	600	8465	1799	0	219	22	29
set1ch	446	420	1055	1346	47304	3872	1	13440	34	141
fixnet6	477	497	1372	1987	22676	96	1	760	34	88
fast0507	484	4922	31409	659	19988	7929	0	7162	19	49
gen	622	360	1303	2076	28874	538	1	8261	22	56
danoint	664	521	3232	1016	20487	7643	1	9207	10	35
misc06	664	755	1957	1480	25175	3417	1	7452	13	43
arki001	769	490	7806	5801	9189	165	1	238	25	39
air04	777	4135	34874	1047	12388	6349	1	3705	7	19
dsbmip	1126	1589	6379	1126	10132	3964	1	2445	7	12
qiu	1192	840	3432	1192	10724	3923	1	2192	7	12
gesa3_o	1200	600	2424	1200	8399	2174	1	1157	7	10
gesa2_o	1248	600	2424	1248	8735	2070	1	1074	4	9
gesa3	1344	600	3744	1344	8063	2948	1	1328	4	8
gesa2	1392	600	3816	1392	9743	2908	1	1009	4	9
rentacar	1426	3014	22293	1426	4190	3127	1	216	4	4
p2756	1653	1614	6940	1653	4958	3235	1	55	1	4
mitre	1657	9590	35014	1657	8281	4927	1	1038	4	7
cap6000	2095	2089	6506	2097	0	0	0	0	1	1
mod011	2332	6776	15699	2332	4663	3794	1	11	1	3
dano3mip	3187	13873	79625	3187	3187	3107	1	0	1	2
seymour	4827	943	30697	4827	4806	4722	1	0	1	2
Total	35466	71311	389351	55141	2165756	643814	120	3367313	35864	50615

Table 6.1: Decomposing Miplib-problems into 2 blocks (Part I)

Example	Best Solutions				Heutistics at Root				Time			
	Lb	Ub	He	No	G	D1	D2	B	LP	Sep	Heu	Tot
mod008	3	**3**	IG	1	3	3	3	6	25%	25%	0%	0.0
p0033	3	**3**	D1	1	3	3	3	14	50%	0%	0%	0.0
flugpl	1	**1**	ID2	1	3	2	1	15	33%	33%	0%	0.0
enigma	10	**10**	IG	1	10	10	10	21	39%	33%	16%	1.9
rgn	5	**5**	IG	1	5	5	5	24	52%	25%	6%	1.0
gt2	11	**11**	IG	1	11	11	11	28	46%	31%	9%	2.6
lseu	6	**6**	D1	1	6	6	6	27	33%	40%	6%	0.1
nw04	18	**18**	D1	1	25	18	18	36	0%	6%	0%	50.7
egout	2	**2**	D1	1	7	2	2	40	50%	37%	0%	0.1
pk1	19	**19**	IG	5	20	22	22	45	20%	31%	17%	286.5
bell5	4	**4**	ID1	2	6	6	6	85	62%	18%	9%	7.5
misc03	43	**43**	IG	37	45	46	46	95	20%	42%	24%	532.2
bell3a	4	**4**	B	6	5	5	5	97	62%	17%	11%	9.8
l152lav	36	**36**	*	485	43	43	43	97	19%	23%	51%	737.3
harp2	17	**17**	D1	1	17	17	17	100	2%	94%	0%	15.7
khb05250	25	**25**	IG	1	25	25	25	100	25%	41%	27%	81.7
p0201	21	**21**	IG	20	30	30	30	113	68%	10%	17%	126.8
stein27	35	56	IG	1	56	56	56	118	33%	43%	7%	3600.2
air03	48	**48**	IG	133	53	58	58	122	55%	23%	19%	1311.6
vpm1	3	**3**	IB	2	6	6	6	129	59%	21%	10%	6.8
vpm2	3	**3**	IB	2	6	6	6	129	60%	20%	11%	6.7
pp08a	8	**8**	D1	1	8	8	8	136	57%	18%	14%	76.9
mod010	51	57	IG	304	61	66	66	146	14%	27%	54%	3600.1
blend2	10	**10**	IG	1	10	10	10	169	62%	24%	2%	1.4
noswot	9	14	IG	9	39	39	39	171	49%	29%	10%	3600.5
10teams	52	90	D1	1	90	90	90	210	47%	33%	15%	3600.6
misc07	69	92	IG	10	106	106	106	223	19%	20%	59%	3627.8
pp08aCUTS	8	**8**	D1	1	8	8	8	246	67%	11%	16%	222.5
p0548	26	47	IG	22	68	68	68	257	62%	15%	15%	3600.6
dcmulti	12	18	ID2	5	25	25	25	272	62%	15%	15%	3601.6
modglob	6	**6**	B	26	10	10	10	287	65%	13%	15%	363.5
fiber	16	22	ID1	22	38	38	38	290	78%	6%	12%	3604.6
rout	17	20	B	12	35	35	35	290	80%	10%	4%	3600.4
p0282	27	36	D1	1	36	36	36	305	68%	10%	17%	3604.6
stein45	19	153	IG	8	157	157	157	331	90%	4%	2%	3616.7
qnet1_o	13	28	D1	1	28	28	28	332	77%	5%	13%	3621.4
qnet1	17	35	D1	1	35	35	35	370	79%	5%	13%	3612.8
air05	19	186	IG	5	189	194	194	408	84%	5%	9%	3604.5
set1ch	11	**11**	D1	1	11	11	11	446	66%	8%	21%	1662.8
fixnet6	13	**13**	D1	1	13	13	13	477	86%	4%	6%	1607.3
fast0507	7	165	IG	6	217	217	217	484	56%	13%	28%	3600.8
gen	8	17	ID1	8	19	19	19	622	73%	3%	20%	3672.4
danoint	5	174	IG	2	189	189	189	664	82%	4%	11%	3611.4
misc06	5	48	ID1	1	48	48	48	664	80%	3%	15%	3711.7
arki001	10	31	D1	1	31	31	31	769	78%	5%	14%	3629.8
air04	4	341	IG	3	367	370	370	777	63%	10%	24%	3725.5
dsbmip	2	57	ID1	1	57	57	57	1126	78%	4%	15%	4253.7
qiu	2	125	ID1	1	125	125	125	1192	73%	4%	20%	3698.1
gesa3_o	2	24	D1	1	24	24	24	1200	85%	2%	10%	4141.2
gesa2_o	2	24	D1	1	24	24	24	1248	88%	2%	6%	4057.6
gesa3	2	120	D1	1	120	120	120	1344	83%	3%	10%	3654.4
gesa2	2	120	ID1	1	120	120	120	1392	82%	2%	12%	5174.6
rentacar	2	227	ID1	1	227	227	227	1426	36%	3%	58%	3662.1
p2756	2	264	ID2	1	264	425	264	1653	91%	3%	3%	6439.9
mitre	1	28	D1	1	28	28	28	1657	35%	4%	57%	4482.4
cap6000	2	**2**	IG	1	2	2	2	2095	10%	18%	1%	170.8
mod011	1	60	ID1	1	60	60	60	2332	9%	3%	82%	3729.6
dano3mip	2	1466	ID1	1	1730	1466	1466	3187	52%	3%	34%	6078.0
seymour	1	1159	ID2	1	1968	1968	1159	4827	7%	5%	66%	9585.0
Total	782	5644		1170	6972	6877	5906	35466	60%	9%	24%	138689.0

Table 6.2: Decomposing `Miplib`-problems into 2 blocks (Part II)

columns, or non-zeros.

Let us first investigate the "dual side" of the results. We observe that we solve very few problems at the root node (only 8 out of 59). The quality of the cuts is in our opinion reasonable, as can be seen from the size of the branch-and-bound tree and the number of LPs solved. It is true, however, that the lower bound improves fast at first while stalling occurs in later stages of the computation although still large numbers of cuts are found and the problem is finished by branch-and-bound. The same behaviour has been reported for similar problems like the node capacitated graph partitioning problem discussed in Ferreira, Martin, de Souza, Weismantel, and Wolsey [1998].

Investigating the "primal side", we see that the greedy heuristic and the two dual heuristics perform very similar, though variant II of the dual heuristics seems to outperform the other two slightly. Bin-packing is either very good (a rather rare event) or a catastrophe, but complements the other heuristics. If we look at the quality of the solutions found at the root node as a measure of the method as a stand-alone decomposition heuristic, the table shows pretty good results. For some instances, as for noswot or fat0507, we detect larger gaps. In fact, we have sometimes observed in longer runs on larger examples that the best solution could steadily be improved and the optimal solution was found late. A reason might be that the heuristics are closely linked to the LP-fixings and essentially always find the same solutions until the branching process forces them strongly into another direction. Nevertheless, in summary the primal heuristics give with very few exceptions a very good performance guarantee, for most of the problems we are within 20% already at the root node.

How decomposable are the mixed integer programming matrices? We see that not all, but many of the larger problems can be brought into bordered block diagonal form. Some like the "bell"- and "vpm"-examples decompose very well. Of course, there are also exceptions like nw04: 18 out of 36 are in the border. The latter is a airline crew scheduling problem, which we did not expect to be decomposable. Anyway, there seems to be potential for the multiple knapsack approach and further research in the direction of finding stronger cutting planes for mixed integer programs, see also Chapter 8.

We also tried to decompose the mixed integer programming matrices into four blocks, having in mind that the decompositions might be used to speed up the solution of the underlying linear programs by exploiting parallelism. The picture of the our results is a bit different here, see Tables 6.3 and 6.4. Since the number of blocks is $\beta = 4$ instead of $\beta = 2$, the integer programming formulation is bigger: The number of variables is doubled, the number of conflicting assignments for two adjacent rows is now 12 instead of 4. Consequently, the LPs tend to be bigger and harder to solve, the percentage of the time spent for solving the LPs goes from 60% up to 72%. Note that $\beta = 4$, on the other hand, halfs the block capacities. This means that it becomes much easier to separate inequalities that have this number as their right-hand side and have a large or combinatorially restrictive support like z-clique, bin-packing, or composition of clique inequalities, compare column BCC in both tables.

On the primal side the results are similar to $\beta = 2$. As expected the number of rows goes up, sometimes significantly like for the "vpm"-examples.

Finally, we want to mention that there is a second application of matrix decomposition to integer programming as a new branching rule. Decomposing the transposed constraint matrix will identify the variables in the border as linking variables. After branching on these variables, the integer program decomposes into smaller integer programs that can be solved independently. Thus, it seems to be a good idea to branch on border variables first. We tried this approach and de-

Example	Presolved			Cutting Planes					B&B	
	Rows	Col	NZs	Init	Cov	2part	BCC	Pool	Nod	Iter
mod008	6	1	6	45	0	0	0	0	1	1
flugpl	15	5	19	105	15	0	0	3	1	2
p0033	14	11	35	95	66	25	10	46	6	17
enigma	21	99	287	120	495	279	125	265	26	67
rgn	24	33	102	152	20565	8437	2237	81333	6506	9278
gt2	28	173	346	109	142	103	10	44	11	21
lseu	27	50	163	152	47	26	2	7	1	3
nw04	36	102	915	344	12	1	139	3	1	3
egout	40	24	76	317	68	4	2	5	1	3
pk1	45	32	105	242	5594	4179	9365	9801	121	419
bell5	85	73	209	415	40418	11209	917	96178	956	1947
misc03	95	133	1801	507	39232	17881	92307	152118	761	1747
bell3a	97	81	235	471	6982	2346	82	9074	31	92
l152lav	97	695	3712	764	56413	31085	57831	293698	576	1173
harp2	100	1225	2450	317	37	19	1	24	1	2
khb05250	100	1275	2574	518	0	0	0	0	1	1
p0201	113	177	1527	529	12003	6915	478	29068	121	243
stein27	118	27	378	940	25574	24840	270	197280	101	291
air03	122	666	5973	1721	10396	8966	22954	34992	206	315
vpm1	129	98	280	1059	104723	14769	2315	189557	3876	6512
vpm2	129	98	280	1059	109973	14448	2322	191469	4081	6748
pp08a	136	120	304	920	7144	1988	54	10216	21	74
mod010	146	1973	8404	1147	16238	12452	1220	51152	106	160
blend2	169	88	1039	1189	0	0	0	0	1	1
noswot	171	50	440	1574	50087	32022	463	1827058	421	574
10teams	210	1600	9600	840	10252	8399	57	18569	41	66
misc07	223	229	8245	1085	473	4407	2852	1740	41	47
pp08aCUTS	246	230	828	1470	11498	3522	48	9534	21	68
p0548	257	250	1009	1306	10203	5793	42	10807	26	48
dcmulti	272	514	1242	1344	23483	8022	89	51090	26	94
modglob	287	354	892	2141	39399	4554	138	58028	51	153
fiber	290	416	1250	1159	15272	6512	54	24934	16	57
rout	290	540	2085	3450	14984	673	35	11806	36	67
p0282	305	168	841	1559	609	583	37	7	1	3
stein45	331	45	1034	2314	6907	6803	21	8712	11	24
qnet1_o	332	800	2400	1572	11241	6228	35	17372	16	38
qnet1	370	924	2806	1186	11815	9543	33	18360	11	35
air05	408	3104	23458	1779	4455	4294	11	608	16	15
set1ch	446	420	1055	3138	21341	5293	49	25448	21	58
fixnet6	477	497	1372	4451	13250	491	27	5637	16	41
fast0507	484	4922	31409	1945	3859	3715	8	758	11	11
gen	622	360	1303	4774	15938	608	21	6374	16	30
danoint	664	521	3232	2696	9956	8984	16	11885	6	17
misc06	664	755	1957	3624	11283	4274	18	8438	11	20
arki001	769	490	7806	12391	7514	406	6	2162	21	15
air04	777	4135	34874	2978	4659	4596	6	524	11	9
dsbmip	1126	1589	6379	3378	12385	7040	12	7832	6	13
qiu	1192	840	3432	3580	9503	7013	8	849	11	11
gesa3_o	1200	600	2424	3600	15598	7637	14	10815	6	15
gesa2_o	1248	600	2424	3744	13727	7038	12	8041	6	13
gesa3	1344	600	3744	4032	12095	6732	10	5585	6	11
gesa2	1392	600	3816	4176	11135	6087	9	4538	6	10
rentacar	1426	3014	22293	4286	6960	5658	5	615	6	7
p2756	1653	1614	6940	4959	4957	4190	4	32	6	5
mitre	1657	9590	35014	4971	8280	5843	6	2481	6	7
cap6000	2095	2089	6506	6287	0	0	0	0	1	1
mod011	2332	6776	15699	6997	6991	6025	3	91	1	4
dano3mip	3187	13873	79625	9561	3187	3153	2	7	1	2
seymour	4827	943	30697	14481	0	0	0	0	1	1
Total	35466	71311	389351	146065	859433	356110	196792	3507070	18424	30710

Table 6.3: Decomposing `Miplib`-problems into 4 blocks (Part I)

Example	Best Solutions				Heutistics at Root				Time			
	Lb	Ub	He	No	G	D1	D2	B	LP	Sep	Heu	Tot
mod008	4	**4**	IG	1	4	4	4	6	40%	0%	0%	0.1
flugpl	4	**4**	IG	1	4	4	4	15	60%	20%	0%	0.1
p0033	5	**5**	IG	1	5	5	5	14	60%	5%	10%	0.2
enigma	11	**11**	IG	1	11	11	11	21	56%	25%	5%	2.0
rgn	10	**10**	IG	1	10	10	10	24	32%	19%	7%	134.8
gt2	11	**11**	D1	1	11	11	11	28	48%	35%	3%	0.9
lseu	7	**7**	D1	1	7	7	7	27	41%	29%	0%	0.2
nw04	25	**25**	IG	1	25	25	25	36	1%	6%	0%	49.3
egout	4	**4**	IG	1	4	4	4	40	61%	19%	4%	0.2
pk1	24	**24**	IG	1	24	24	24	45	53%	28%	5%	52.4
bell5	10	**10**	IG	22	13	13	13	85	78%	9%	5%	623.6
misc03	54	**54**	IG	3	59	59	59	95	34%	41%	8%	1329.0
bell3a	7	**7**	B	11	17	17	17	97	83%	8%	4%	82.9
l152lav	43	49	IG	19	56	63	63	97	76%	15%	4%	3600.9
harp2	18	**18**	D1	1	20	18	18	100	2%	95%	0%	15.4
khb05250	25	**25**	D1	1	26	25	25	100	3%	95%	0%	12.3
p0201	30	**30**	IB	36	39	39	39	113	90%	4%	3%	650.5
stein27	28	79	IG	6	83	85	85	118	94%	3%	0%	3600.6
air03	51	72	IG	42	82	82	82	122	83%	13%	2%	3608.4
vpm1	10	**10**	B	42	13	13	13	129	73%	11%	7%	2529.1
vpm2	10	**10**	B	45	13	13	13	129	73%	11%	6%	2556.3
pp08a	8	**8**	D1	1	8	8	8	136	89%	5%	3%	184.5
mod010	39	82	IG	17	84	86	86	146	93%	3%	2%	3604.3
blend2	10	**10**	IG	1	10	10	10	169	77%	17%	0%	3.3
noswot	7	19	IG	36	39	39	39	171	66%	19%	3%	3613.6
10teams	46	90	D1	1	90	90	90	210	85%	5%	9%	3666.7
misc07	81	146	IG	6	150	160	160	223	78%	11%	9%	3688.3
pp08aCUTS	8	**8**	D1	1	8	8	8	246	88%	4%	5%	680.9
p0548	8	75	IG	5	80	83	80	257	97%	1%	0%	3620.8
dcmulti	8	31	D2	1	31	47	31	272	96%	2%	0%	3622.5
modglob	8	16	D1	7	18	18	18	287	93%	3%	2%	3606.1
fiber	6	38	D1	1	38	38	38	290	96%	1%	1%	3628.4
rout	11	22	IG	2	35	35	35	290	97%	2%	0%	3709.5
p0282	36	**36**	D1	1	36	36	36	305	29%	25%	42%	46.7
stein45	6	232	IG	2	244	244	244	331	96%	2%	0%	3724.4
qnet1_o	6	28	D1	1	28	28	28	332	95%	1%	2%	3768.7
qnet1	6	35	D1	1	35	35	35	370	94%	1%	3%	3985.3
air05	72	285	IG	2	293	297	293	408	90%	5%	3%	3803.8
set1ch	7	11	D1	1	11	11	11	446	90%	2%	5%	3610.2
fixnet6	6	16	D1	1	16	16	16	477	95%	1%	1%	3602.7
fast0507	36	301	IG	2	337	337	337	484	93%	3%	1%	4247.4
gen	6	33	ID1	2	41	41	41	622	86%	3%	9%	3689.6
danoint	4	214	IG	1	214	223	223	664	92%	2%	4%	4281.3
misc06	5	90	D1	1	90	90	90	664	93%	2%	4%	4136.7
arki001	24	31	D1	1	31	31	31	769	91%	4%	3%	4011.7
air04	38	526	IG	2	536	558	558	777	85%	4%	8%	3963.1
dsbmip	3	228	D1	1	228	228	228	1126	69%	5%	23%	4157.4
qiu	6	132	ID2	1	132	207	132	1192	82%	3%	12%	5094.8
gesa3_o	4	45	D1	1	45	45	45	1200	79%	5%	12%	3812.2
gesa2_o	4	48	D1	1	48	48	48	1248	81%	4%	12%	4138.1
gesa3	3	160	D1	1	160	160	160	1344	80%	4%	12%	4226.6
gesa2	3	168	D1	1	168	168	168	1392	80%	5%	11%	3667.0
rentacar	10	302	ID1	1	302	302	302	1426	59%	4%	33%	4335.3
p2756	2	462	ID2	1	462	688	462	1653	77%	8%	11%	3867.1
mitre	1	106	ID1	2	131	135	131	1657	43%	5%	47%	4082.2
cap6000	2	**2**	IG	1	2	2	2	2095	8%	74%	0%	730.4
mod011	3	105	D1	1	105	105	105	2332	33%	5%	54%	4038.7
dano3mip	2	1514	ID2	1	1868	1681	1514	3187	14%	2%	76%	9967.3
seymour	0	2351	ID2	1	2965	2965	2351	4827	3%	2%	78%	11846.3
Total	926	8475		349	9645	9835	8726	35466	72%	5%	18%	163313.3

Table 6.4: Decomposing `Miplib`-problems into 4 blocks (Part II)

composed the transposed integer programming matrices. As a surprise it turned out that for the examples that decompose reasonably well almost always all border variables are continuous variables. The only exceptions are the two "bell"-examples and noswot. We run SIP for these three examples and branched on one of the border variables first whenever they have been fractional. The improvement we obtained were only marginal, possible reasons are that the "bell"-examples contain only one or two integer variables in the border, and for noswot the value of the LP relaxation already coincides with the value of the integer optimal solution and the difficulty of this problem lies in finding this solution.

6.5.2 Steiner Tree Packing Problems

We also tried to test our branch-and-cut algorithm on Steiner tree packing problems as described in Chapter 3. To get reasonable test problems we are faced with the difficulty that the integer programming formulation (3.1) contains an exponential number of Steiner cut inequalities. Although there are much less necessary at the end to prove optimality of a solution, it is not clear from the beginning which ones they are. To get around this problem we looked for examples where after adding only Steiner cut inequalities and no joint inequalities the root LP yields an integer solution. We found four reasonably small instances that have this property. Table 6.5 summarizes the data.

Example	h	w	N	Steiner Cuts per Net								Border
				1	2	3	4	5	6	7	8	
g353	3	5	3	23	12	24						22
g444	4	4	4	27	14	17	30					23
d677	6	7	7	52	36	39	52	31	26	32		71
d688	6	8	8	29	48	42	40	38	27	34	23	82

Table 6.5: Steiner tree packing problems: Data

Column 1 gives the name of the problem, Columns 2 and 3 the height and width of the underlying grid graph. The number of nets is shown in Column 4. Columns 5 through 12 present the number of Steiner cut inequalities in the root LP for each single net. The last column gives the number of capacity constraints. Note that due to the fixing of certain variables in the initialization phase of our branch-and-cut algorithm for the Steiner tree packing problem, see Chapter 3, this number might be less than the number of edges ($= (h-1)\cdot w + (w-1)\cdot h$). If we use for β the number of nets and for the block capacity κ the maximal number of Steiner cut inequalities per net, an obvious solution to the matrix decomposition problem is to put all capacity constraints in the border.

Example	Presolved			Cutting Planes					B&B	
	Rows	Col	NZs	Init	Cov	2part	BCC	Pool	Nod	Iter
g353	81	48	276	336	60476	16959	830	149368	645	1586
g444	111	36	226	718	9753	3049	142	14625	86	203
d677	339	174	1129	3721	12229	6759	50	35288	25	53
d688	363	222	1241	4205	13822	8105	52	59919	19	54
Total	894	480	2872	8980	96280	34872	1074	259200	775	1896

Table 6.6: Decomposition of Steiner tree packing matrices (Part I)

Example	Best Solutions				Heuristics at Root				Time			
	Lb	Ub	He	No	G	D1	D2	B	LP	Sep	Heu	Tot
g353	20	**20**	IG	89	26	26	26	81	71%	13%	6%	518.0
g444	10	**10**	D1	53	17	18	18	111	77%	12%	4%	125.0
d677	7	75	IG	3	105	105	105	339	97%	1%	0%	10986.6
d688	8	80	IG	3	123	123	123	363	97%	1%	0%	11194.2
Total	45	185		148	271	272	272	894	97%	1%	0%	22823.8

Table 6.7: Decomposition of Steiner tree packing matrices (Part II)

Tables 6.6 and 6.7 show the results we obtain with our branch-and-cut algorithm for the associated matrix decomposition problems. We see that for the two smaller examples the solution is even better than the "natural" decomposition. The reason is that some of the capacity constraints turn out to be trivial inequalities due to variable fixings right at the beginning of our branch-and-cut algorithm for the Steiner tree packing problem. Unfortunately, we are far from solving the two larger examples. The difficulty of the matrices from Steiner tree packing problems lies in the fact that the capacity constraints that are supposed to be in the border are very sparse compared to the Steiner cut inequalities. For instance, the capacity constraints of example d677 have on average 2.6 non-zero entries (the maximal number of non-zeros is 5), whereas the average density of the Steiner cut inequalities for this example is 3.6 with a maximum of 13. The heuristics and the LP solutions, however, tend to put the rows into the border that have the most non-zero entries.

Chapter 7

Parallelizing the Dual Simplex Method

7.1 Introduction

In Chapters 2, 3, and 4 we have seen that a significant amount of time is spent for the solution of the underlying linear programs. For example, in case of the Steiner tree packing problem up to 90% and more of the total time is due to the solution of the linear programs. Thus, in order to solve real-world problems of large scale (as they appear in Chapters 2, 3, and 4) it is necessary to speed-up the solution process of the linear programs. That is to speed-up the dual simplex algorithm, the method that is used within a branch-and-cut algorithm (see Appendix B). Linear programs resulting from integer programs with block structure give two opportunities to speed-up the dual simplex method (see the description of the dual simplex method in the next section): First, the LU-factorization, since each block can be factorized independent of the other blocks and only the linking constraints must be factorized sequentially thereafter. Second, all column based operation like pricing can be parallelized. The latter, of course, only pays if a significant amount if time is spent in these operations. Good candidates here are linear programs that result from integer programming applications, particularly those that employ "column-generation", or linear programs that arise from integer programs with block structure. For example, in case of the multiple knapsack problem, the number of rows in the initial integer programming formulation, see (2.1), is $|N| + |M|$, whereas the number of variables is $|N| \cdot |M|$. Thus, the ratio of number of variables to number of rows grows quadratic, improving the potential for parallelism with increasing problem sizes. The same might also hold for the Steiner tree packing problem and the PIPE problem studied in Chapters 3 and 4, respectively. Of course, it is difficult to estimate the number of cuts (i) that are necessary in the integer programming formulations (3.1) and (4.1), but with an increasing number of nets (demands) N the percentage of "local" nets (demands) increases. For these "local" nets (demands) the number of necessary cuts is usually sublinear. Thus, also in case of the Steiner tree packing problem and the PIPE problem the number of variables $N \cdot |E|$ might grow quadratic with the number of rows $O(N + |E|)$. Hence, for all real-world problems discussed in Part I a significant amount of time is spent in the column operations of the dual simplex method.

In this chapter we investigate parallelizing the CPLEX[1] implementation of the dual simplex algorithm. As suggested by the above discussion, we will concentrate our efforts on the pricing and other column-based steps in the dual simplex method.

[1]CPLEX is a registered trademark of ILOG

We examine three different parallel implementations and test them on different platforms. Our results will show that the best parallel implementation is superior to the sequential by a factor of at least 1.5 already if the ratio of number of variables to number of rows is at least 10. Thus, the parallel dual simplex algorithm might indeed help to speed-up the solution process of integer programs with block structure that result from real-world applications of large scale.

In the next section we begin with an outline of the steps of the dual simplex method followed by profiling results and a more detailed discussion of our parallelization. For the profiles we have selected four test problems with a range of aspect ratios. The ensuing sections describe our various implementations. We start with an implementation using PVM (Beguelin, Dongarra, Geist, Jiang, Manchek, and Sunderam [1994]), followed by one using System V shared-memory constructs, and conclude with by far the most successful implementation based upon the PowerC extension of the C programming language. Finally, we give computational results. These results use an extensive set of test problems, statistic for which appear in Appendix D.

Other work on the parallelization of the simplex algorithm includes the following. Helgason, Kennington, and Zaki [1988] present a parallelization of the simplex method based on a quadrant interlocking factorization, but no computational results are given. Eckstein, Boduroglu, Polymenakos, and Goldfarb [1995] investigate an implementation of a more practical revised simplex method, but the assumption is made that the constraint matrices are dense, a rare occurrence in practice, see the tables about problem statistics in Appendix D. In Wunderling [1996] a parallel implementation of the simplex algorithm for sparse linear systems is described where good speed-ups could be obtained for problems with a high ratio of variables to constraints. Parallelizing the LU factorization is a topic of its own and has highly been investigated in computational linear algebra, see the books Duff, Erisman, and Reid [1986], Kumar, Grama, Gupta, and Karypis [1994], and Gallivan, Heath, Ng, Ortega, Peyton, Plemmons, Romine, Sameh, and Voigt [1990] for surveys and further references. Finally, Barr and Hickman [1994], Chang, Engquist, Finkel, and Meyer [1988], and Peters [1990] discuss the parallelization of the network simplex method. In contrast to the dual simplex algorithm, the network simplex method exploits specific problem structure. An indirect result of this fact is that the major part of computational effort is spent in pricing (for large scale problems typically more than 90%). All three papers focus on the merits of parallelizing this pricing step.

7.2 Dual Simplex Algorithms

We suppose the reader to be familiar with the basic terms of linear programming. For a good introduction to linear programming see Chvátal [1983] or Padberg [1995].

Consider a *linear program (LP)* in the following *standard form*:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array} \tag{7.1}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$. Note that most practical LPs have non-trivial bounds on some variables; however, for purposes of this discussion it will suffice to consider problems in the form (7.1).

The dual of (7.1) is

$$\begin{array}{ll} \max & b^T \pi \\ \text{s.t.} & A^T \pi \leq c \end{array} \tag{7.2}$$

Adding slacks yields

$$\begin{array}{lrl} \max & b^T\pi & \\ \text{s.t.} & A^T\pi + d & = c \\ & d & \geq 0 \end{array} \tag{7.3}$$

A *basis* for (7.1) is an ordered subset $B = (B_1, \ldots, B_m)$ of $\{1, \ldots, n\}$ such that $|B| = m$ and $\mathbf{B} := A_B$ is non-singular. B is *dual feasible* if $c_N - A_N^T \mathbf{B}^{-T} c_B \geq 0$, where $N = \{1, \ldots, n\} \backslash B$ and $\mathbf{B}^{-T}$ abbreviates $(\mathbf{B}^{-1})^T$.

Algorithm 7.2.1 *A generic iteration of the* standard dual simplex algorithm *for the linear program (7.1).*

Input: *A dual feasible basis* B, $\bar{d}_N = c_N - A_N^T \mathbf{B}^{-T} c_B$ *and* $\bar{x}_B = \mathbf{B}^{-1} b$.

(1) If $\bar{x}_B \geq 0$, B *is optimal* – **Stop**;
otherwise, let
$i = \operatorname{argmin}\{\bar{x}_{B_k} : k = 1, \ldots, m\}$.
d_{B_i} *is the* entering variable.
(2) Solve $\mathbf{B}^T z = e_i$. *Compute* $\alpha_N = -A_N^T z$.
(3) Ratio Test.
If $\alpha_N \leq 0$, *(7.1) is infeasible* – **Stop**;
otherwise, let
$j = \operatorname{argmin}\{\bar{d}_k/\alpha_k : \alpha_k > 0, k \in N\}$.
d_j *is the* leaving variable.
(4) Solve $\mathbf{B}y = A_j$.
(5) Set $B_i = j$. *Update* $\bar{x}_B$ *(using* y*) and* $\bar{d}_N$ *(using* z*).*

Remarks:

1. Note that in Step (1) we refer to the choice of the *entering* dual variable d_{B_i}, while most textbooks refer to the corresponding *leaving* primal variable x_{B_i}. A similar remark applies to Step (3).
2. For all dual simplex algorithms, the efficient computation of $z^T A_N$ is crucial. This computation is implemented by storing A_N row-wise so that zero elements in z need be examined only once.
3. To improve stability, the Ratio Test (Step (3)) is applied in several passes, using an idea of Harris [1973]. First, the ratios

$$r_k = \begin{cases} \bar{d}_k/\alpha_k & \text{if } \alpha_k > 0 \text{ and} \\ +\infty & \text{otherwise,} \end{cases} \tag{7.4}$$

are computed for each $k \in N$. Using these ratios, we compute

$$t = \min\{r_k + \epsilon/\alpha_k : k \in N\}, \tag{7.5}$$

where $\epsilon > 0$ is the *optimality tolerance*, by default 10^{-6}. Finally, we compute the actual leaving variable using the formula

$$j = \operatorname{argmax}\{\alpha_k : r_k \leq t\}. \tag{7.6}$$

Note, since $\epsilon > 0$, it is possible for some of the d_k to be negative, and hence that r_j is negative. In that case, depending upon the magnitude of r_j, we may *shift* c_j to some value at least $c_j + |d_j|$, and then repeat the calculation of t and j employing the new r_j. (See Gill, Murray, Saunders, and Wright [1989] for a discussion of the approach that suggested this *shifting*. The details of how these shifts are removed have no effect on our implementation and are omitted.)

4. In order to solve the two linear systems in the above algorithm (see Steps (2) and (4)), we keep an updated LU-factorization of $\mathbf{B}$, using the so-called Forrest-Tomlin update, see Forrest and Tomlin [1972]. For most models, a new factorization is computed once every 100 iterations. These computations may be considered part of Step (5).

Steepest Edge

There are three different dual algorithms implemented in CPLEX: The standard algorithm, described above, and two *steepest-edge* variants. The default algorithm is steepest-edge.

Several steepest-edge alternatives are proposed in Forrest and Goldfarb [1992]. These algorithms replace the rule for selecting the index of the entering variable d_{B_i} by

$$i = \operatorname{argmin}\{\bar{x}_{B_k}/\eta_k : \ k = 1, \ldots, m\},$$

where the η_k are the *steepest-edge norms*. The alternative used in our tests corresponds to the choice

$$\text{(SE)} \quad \eta_k = \sqrt{(e_k^T \mathbf{B}^{-1})(e_k^T \mathbf{B}^{-1})^T}.$$

While it is too expensive to explicitly compute all η_k at each iteration, there are efficient update formulas. Letting $\{\eta_1, ..., \eta_m\}$ be the values of the norms at the start of an iteration, the values at the start of the next iteration for (SE), $\bar{\eta}_k$, are given by the formula

$$\text{(SE norm update)} \quad \bar{\eta}_k^2 = \eta_k^2 - 2(\frac{y_k}{y_i})e_k^T \mathbf{B}^{-1} z + (\frac{y_k}{y_i})^2 z^T z \quad (k \neq i),$$

where y and z are as in the statement of the standard dual simplex algorithm. Note that the implementation of this formula requires the solution of one extra linear system per iteration, the one used to compute $\mathbf{B}^{-1}z$. As suggested in Forrest and Goldfarb [1992], this second "FTRAN" can be solved simultaneously with the linear system in Step (4), thus requiring only a single traversal of the updated LU-factorization of $\mathbf{B}$.

The default dual in CPLEX uses the (SE) norms with the approximate starting values $\eta_k = 1$ for all k. This choice corresponds to the assumption that most variables in the initial basis will be slacks or artificials. See Forrest and Goldfarb [1992] for a detailed discussion.

Summary

In the sections that follow we discuss three different parallel implementations of the (SE) variant of the standard dual simplex method: One using PVM, one using general-purpose System V shared-memory constructs, and one using the PowerC extension of C on an Silicon Graphics multi-processor. In Section 7.3, we begin by outlining the basic plan for the PVM and "System V" approaches. Each of these requires some explicit form of data distribution. The PowerC version requires no such data distribution.

To set the stage for the ensuing sections, we close this section with a discussion of which steps in the dual simplex can be parallelized and give four profiles for runs on an SGI Power Challenge using the sequential version of CPLEX. The problem characteristics for the problems selected are given in Table D.3 in Appendix D.

Since the steps that we have chosen to parallelize (as discussed below) are with one exception in Section 7.6 all column based, it is apparent that the percentage of parallel work will increase as the aspect ratio of the selected LP increases. The examples we have chosen demonstrate this fact quite clearly.

In the discussions that follow, we make use of the following designations, classifying the various parts of the algorithm:

Designation	Description
Enter	Step (1).
BTRAN	Solution of $\mathbf{B}^T z = e_i$ (Step (2)).
Pricing	Computation of $\alpha_N = -A_N^T z$ (Step (2)).
Ratio	Computation of t (Step (3) and (7.5)).
Pivot	Computation of j and shifting, if necessary (Step (3) and (7.6)).
FTRAN	Solutions of $\mathbf{B}y = A_j$ and $\mathbf{B}w = z$.
Factor	Factorization and factorization update (Step (5)).
Update-d	Update of $\bar{d}_N$.
Update-x	Update of $\bar{x}_B$ and $\bar{\eta}$.
Misc	All other work.

The Pricing, Ratio, Pivot, Update-d, and Update-x steps offer clear opportunities for parallelism. We have chosen to concentrate on the first four of these. For most practical LPs, Step *Update-x*, seems unlikely to consume a significant part of the total computation time: In typical LPs, the number of rows is smaller than the number of columns, usually by a multiple of at least 2 to 3, often by much more. Indeed, we *did* test this hypothesis while testing our PowerC implementation, and found that a parallel version of Update-x was at best of marginal value, and in some cases actually degraded performance.

Of the remaining steps, BTRAN and FTRAN are highly recursive and well known to be very difficult to parallelize, especially given the fact that the LU-factorization of the basis matrix $\mathbf{B}$ changes by a rank-1 update at each iteration. Even the "obvious parallelism", afforded by solving each of the two systems in FTRAN on separate processors, is difficult to exploit, see the discussion in Section 7.6. Finally, the LU-factorization has a clear potential to be parallelized, in particular if the underlying LP has block structure. However, the problem of parallelization of the LU-factorization is largely independent of the simplex method itself, and we have chosen not to investigate it here. See Helgason, Kennington, and Zaki [1988], Davis and Yewr [1990], Duff, Erisman, and Reid [1986], Kumar, Grama, Gupta, and Karypis [1994], and Gallivan, Heath, Ng, Ortega, Peyton, Plemmons, Romine, Sameh, and Voigt [1990] for a further discussion of this problem.

Table 7.1 shows CPLEX profile statistics. The entries show, for four examples, the time in percentage spent in the various parts of the sequential algorithm. The rows in bold face give the steps that will be parallelized in this chapter. The last row *% Parallel* (expressed in percentage of the total running time) summarizes these parts. Thus, for cre_b we expect a reduction to at most 44.3% + $\frac{55.7\%}{p}$ of the sequential time, when using p processors, whereas for aa300000 almost linear speed-ups may be obtained.

Algorithmic Step	% of Total Computation Time pilots	cre_b	roadnet	aa300000
Enter	2.1	5.5	0.2	0.1
BTRAN	15.0	11.5	1.5	0.5
Pricing	**15.3**	**33.1**	**57.2**	**65.4**
Ratio	**5.3**	**15.6**	**22.7**	**20.4**
Pivot	**2.3**	**3.9**	**6.9**	**4.4**
FTRAN	31.2	20.5	3.3	1.1
Factor	20.3	3.7	1.2	0.4
Update-d	**1.1**	**3.1**	**5.2**	**7.4**
Update-x	2.5	0.6	0.6	0.2
Misc	4.9	2.5	1.2	0.1
Total	100.0	100.0	100.0	100.0
% Parallel	**24.0**	**55.7**	**92.0**	**97.6**

Table 7.1: CPLEX profiles for some LP problems

7.3 Outline of the Data Distributed Implementation

In this section we discuss our data distributed implementations of the (SE) version of the standard dual simplex method. The parallel model we use is master/slave with one master and (potentially) several slaves. We call the master the *boss* and the slaves *workers*. The *boss* keeps the basis, and each processor, including the *boss*, gets a subset of columns. Each column must belong to exactly one processor. All computations directly related to the basis are done sequentially, by the *boss*. The other steps can be executed in parallel: Pricing, Ratio, Pivot, and Update-d.

Algorithm 7.3.1 *A* parallel iteration *of the* dual simplex algorithm.

Input: *A dual feasible basis* B, $\bar{d}_N = c_N - A_N^T \mathbf{B}^{-T} c_B$ *and* $\bar{x}_B = \mathbf{B}^{-1} b$.

(1) Enter.
If $\bar{x}_B \geq 0$, *B is optimal* – **Stop**; *otherwise, let*
$i = \text{argmin}\{\bar{x}_{B_k} : k = 1, \ldots, m\}$.
d_{B_i} *is the entering variable.*

(2) BTRAN.
Solve $\mathbf{B}^T z = e_i$.

(3) **Com(z).**
The boss *sends the vector* z *to the* workers.

(4) **Pricing.**
Each processor computes its part of $\alpha_N = -A_N^T z$.

(5) **Com(α).**
The workers *inform the* boss *whether their parts of* α_N *are non-positive.*

(6) Unboundedness Test.
If $\alpha_N \leq 0$, *(7.1) is infeasible* – **Stop**.

(7) **Ratio.**
The processors compute their t-values, see (7.5).

(8) **Com(t).**
The workers *send their* t *to the* boss.
The boss *determines the global* t *and sends it to the* workers.

(9) **Pivot.**
Each processor determines j as outlined in (7.6).

(10) **Com(p).**
The workers *send their* pivot element $|\alpha_j|$ *to the* boss.

(11) Pivot Selection.
The boss *determines the best pivot and corresponding j and determines if it is "acceptable". If it is rejected, the objective-function coefficient for j is shifted and all processors go back to* Ratio[2].

(12) **Com(j).**
If the pivot element is accepted, the boss *informs the "winning"* worker *to send its column. d_j is the leaving variable.*

(13) FTRAN.
Solve $\mathbf{B}y = A_j$ *and* $\mathbf{B}w = z$.

(14) Factor.
Factorization and its update.

(15) **Com(update).**
The boss *sends information to the* workers *for the update, including the leaving and entering variable.*

(16) Update-x.
Set $B_i = j$. *Update* $\bar{x}_B$ *(using y).*

(17) **Update-d.**
Update $\bar{d}_N$ *(using z).*

Algorithm 7.3.1 outlines a typical iteration of the parallel dual simplex. The steps that do not appear in bold face were described in the previous section in Algorithm 7.2.1 and are sequentially performed by the *boss*. The first new step is the communication of the z vector, **Com(z)**, from the *boss* to the *workers*. For the infeasibility test (see Step (3) of the dual simplex algorithm) the *workers* inform the *boss* in **Com(α)** whether their part of α_N satisfies $\alpha_N \leq 0$.

The steps Ratio, **Com(t)**, Pivot, **Com(p)**, and Pivot Selection must then be performed iteratively until the pivot has been accepted. In **Com(t)** the global t, see (7.5), is determined and distributed among the processors. This involves two communications steps. After the *workers* send their pivot element in **Com(p)** to the *boss* (another communication step) the *boss* decides on the acceptance of the pivot. It is rejected, the *boss* informs the *workers* to return to Ratio. Thus, the total number of communication steps until the pivot element is accepted is $4 \cdot (\text{number of rejected pivots}) + 3$.

After the pivot element has been accepted, the *boss* informs the "winning" *worker* to send the entering column (two communication steps). The data in **Com(update)** includes the leaving variable and data for updating the reduced costs. This information is collected at different points within the sequential code, resulting in at most two communication steps. Table 7.2 gives a diagram of Algorithm 7.3.1 and shows where communication steps occur and which steps are performed in parallel.

In view of the profile statistics given in Table 7.1 in the previous section, and of the fact that Enter, BTRAN, FTRAN and Factor will all be executed on a single processor (the *boss*), it is plain that we cannot expect significant performance improvements unless the ratio of variables to constraints in a given LP is large. Indeed, our first thought was not only to enforce this requirement, but to concentrate

[2]The complete test for pivot acceptability is much more complicated than indicated here, but the basic structure of the algorithmic response is essentially as indicated.

Step	*boss*		*worker*
Enter	$*$		
BTRAN	$*$		
Com(z)		$\xrightarrow{z}$	
Pricing	$*$		$*$
Ratio	$*$		$*$
Com(α)		$\xleftarrow{\alpha}$	
Com(t)		$\xleftrightarrow{t}$	
Pivot	$*$		$*$
Com(p)		$\xleftarrow{\lvert\alpha_j\rvert}$	
Pivot Selection	$*$		
Com(j)		$\xleftrightarrow{j}$	
FTRAN	$*$		
Factor	$*$		
Com(update)		$\xrightarrow{update}$	
Update-d	$*$		$*$
Update-x	$*$		

Table 7.2: The arrows in this table indicate where communication between the *boss* and the *workers* must occur, with directions indicating the direction of data flow. An asterisk marks where a task is performed.

on problems for which the total memory requirements were so large that they exceeded the memory available on a single processor. Thus, we began by considering possibly heterogeneous networks of workstations connected by a local area network. As communication software we used PVM.

7.4 An Implementation Using PVM

PVM (Parallel Virtual Machine) is a general purpose software package that permits a network of heterogeneous Unix computers to be used as a single distributed-memory parallel computer, called a virtual machine. PVM provides tools to automatically initiate tasks on a virtual machine and allows tasks to communicate and synchronize[3].

Our first implementation was in one-to-one correspondence with the sequential code. Thus, the *boss* immediately sent a request to the *workers* whenever some particular information was needed. Where possible, the *boss* then performed the same operations on its set of columns, thereafter gathering the answers from the *workers*. Assuming that the first selected pivot was accepted, this approach led to from 6 to 9 communication steps per iteration, depending on whether the entering and/or leaving column belonged to the *workers*. The data was partitioned in our initial implementation by distributing the columns equally among the processors.

We tested this first parallel implementation, carried out on the `Netlib` problems, on a cluster of two workstations. The *boss* was run on a SUN S20-TX61 and the one *worker* on a SUN 4/10-41. The two workstations were connected by a 10 Mb/s (megabits per second) Ethernet. The sequential code was run on the SUN S20-TX61. Columns 2 and 3 of Table 7.3 show the solution times and the number

[3] PVM is public domain and accessible over anonymous ftp via netlib2.cs.utk.edu. For details on PVM, see Beguelin, Dongarra, Geist, Jiang, Manchek, and Sunderam [1994]. In our implementation we used PVM Version 3.3.7.

of iterations needed by the sequential code, Columns 4 and 5 the corresponding numbers for the initial parallel implementation. All solution times given are real (wall-clock) times in seconds, unless otherwise noted, and are for the reduced models obtained by applying the default CPLEX presolve procedures. The times do not include reading and presolving. Results for larger problems are presented later. Note that the parallel version was approximately 3.3 times slower than the sequential version! Due to the nature of the Ethernet, this was not unexpected. But not all of this excess time was due to communication costs, which suggested the following improvements.

1. In Com(p) each *worker* sends not only the pivot element, but simultaneously the corresponding column. This modification saves Com(j), since the *boss* no longer needs to inform the "winning" *worker* to send a column.

2. The pivot selection strategy was changed to reduce the number of communication steps. Each processor determines its own t and performs the steps Ratio, Pivot and Pivot Selection (including shifting) independently of the other processors. The *workers* then send their selected pivots and t values to the *boss*, which makes the final selection. This procedure reduces the number of communication steps of steps Ratio through Pivot Selection, Steps (7) - (11), from $4 \cdot$ (number of rejected pivots) + 3 to 3.

3. The information for the infeasibility test Com(α) can be sent in Com(p). In case infeasibility is detected, the pivot computation is wasted work, but such occurrences are rare.

4. All relevant information for the *workers*' update is already available before FTRAN. Note that the *workers* need only know the entering and leaving column and the result from the Ratio Test in order to update the reduced costs. Thus, only one communication step after Pivot Selection is needed for the update.

5. PVM offers different settings to accelerate message passing for homogeneous networks. We make use of these options where applicable.

6. Load balancing was (potentially) improved as follows: Instead of distributing columns based simply upon the number of columns, we distributed the matrix non-zeros in as nearly equal numbers as possible over all processors.

Columns 6 and 7 of Table 7.3 show the results on the `Netlib` problems after implementing the above improvements. For a typical simplex iteration, the number of communication steps was reduced to three: the *boss* sends z, the *workers* send their pivots and corresponding columns, and the *boss* sends information for the update.

Example	Sequential		2 processors (initial)		2 processors (improved)	
	Time	Iterations	Time	Iterations	Time	Iterations
`Netlib`	3877.8	130962	12784.8	137435	7736.5	142447

Table 7.3: `Netlib` results on a local area network

Based upon Table 7.3, the implementation of 1.-6. improves computational times by a factor of 1.6, even though increasing the number of iterations slightly. However, the performance of the parallel code is still significantly worse than that of the sequential code. One reason is certainly the nature of the `Netlib` problems. Most are either very small or have a small number of columns relative to the number of rows. Table 7.4 gives corresponding results for a test set where the ratio of columns to rows was more favorable, see the problem statistics in Table D.3 of Appendix D.

Example	Sequential		2 processors	
	Time	Iterations	Time	Iterations
0321.4	9170.1	21481	7192.0	20178
cre_b	614.5	11121	836.1	13219
nw16	120.7	313	83.1	313
osa030	645.8	2927	515.4	3231
roadnet	864.7	4578	609.6	4644

Table 7.4: Larger models on a local area network

The results are significantly better. With the exception of cre_b, the parallel times are between 20% (for osa030) and 37% (for nw16) faster, though, again largely due to communication costs, still not close to equaling linear speed-up. Our measurements indicated that communication costs amounted to between 30% (for osa030) and 40% (for cre_b) of the total time. Since communication was taking place over Ethernet, we decided to test our code on two additional parallel machines where communication did not use Ethernet, a SUN S20-502 with 160 MB of RAM memory and an IBM SP2 with eight processors (each a 66 MHz thin-node with 128 MB of RAM). The nodes of the SP2 were interconnected by a high speed network running in TCP/IP mode.

Example	Sequential		2 processors	
	Time	Iterations	Time	Iterations
`Netlib`	4621.2	130962	6931.1	142447
0321.4	9518.3	21481	8261.1	20178
cre_b	650.5	11121	769.4	13219
nw16	99.6	313	78.4	313
osa030	556.3	2927	502.1	3231
roadnet	801.0	4578	652.5	4644

Table 7.5: Larger models on a SUN S20-502

The results on the SUN S20-502 were unexpectedly bad, worse than those using Ethernet. We will come to possible reasons for this behavior later. The results on the SP2 were much better (with the exception of cre_b) and seem to confirm our conclusions concerning the limitations of Ethernet.

Example	Sequential		2 processors		4 processors	
	Time	Iterations	Time	Iterations	Time	Iterations
`Netlib`	2140.9	130054	5026.9	143348	not run	not run
0321.4	5153.7	24474	3624.6	26094	2379.7	21954
cre_b	390.2	11669	399.8	11669	458.9	10915
nw16	94.0	412	50.4	412	30.4	412
osa030	321.3	2804	191.8	2804	152.7	2836
roadnet	407.3	4354	235.5	4335	182.4	4349

Table 7.6: Larger models on an SP2

To summarize, there seems little hope of achieving good parallel performance on a general set of test problems using PVM and a distributed-memory model. Indeed, it is our feeling that this conclusion is valid independent of PVM. Such a result is not unexpected. However, the distributed memory code is not without applications as illustrated by the final table of this section.

Example	Time	Iterations
aa6000000	10315.8	10588
us01	59.4	249

Table 7.7: Large airline models on an SP2 using all 8 nodes.

The two examples in Table 7.7 did not fit onto a single node of the machine being used, so we could not compare the numbers to sequential times. However, the CPU-time spent on the *boss* was 9332.9 sec. (90.5% of the real time) for aa6000000 and 52.5 sec. (= 88.5% of the real time) for us01. Time measurements for the smaller examples in Table 7.6 confirm that about 10% went for communication.

In closing this section, we note that one of the biggest limitations of PVM is directly related to its portability. The generality of PVM means that transmitted data usually must be passed through different interfaces and thereby often packed, unpacked, encoded, decoded, etc. For multiprocessors like the SUN S20-502 or the Power Challenge (see Section 7.6), this work is unnecessary.

7.5 A Shared Memory Implementation

Based upon our results using PVM we decided to investigate the use of general-purpose, UNIX System V shared-memory constructs. We restricted our choice to System V mainly because it provides high portability. Possible candidates for interprocess communication (IPC) on a single computer system are *pipes*, *FIFOs*, *message queues*, and *shared memory* in conjunction with *semaphores* (for an excellent description of these methods see Stevens [1990]). We looked at the performance of these four types of IPC by sending data of different sizes between two processors. It turned out that the shared memory/semaphore version was the fastest (see also Stevens [1990], page 683). *Shared Memory* allows two or more processes to share a certain memory segment. The access to such a shared memory segment is controlled by *semaphores*. Semaphores are a synchronization primitive. They are intended to let multiple processors synchronize their operations, in our case the access to shared memory segments. There are different system calls available that create, open, give access, modify or remove shared memory segments and semaphores. For a description of these functions, see the man pages of Unix System V or Stevens [1990].

We implemented our shared memory version in the following way: We have one shared memory segment for sending data from the *boss* to the *workers*. This segment can be viewed as a buffer of appropriate size. All the data to be sent to the *workers* is copied into this buffer by the *boss* and read by the *workers*. The *workers* use the first four bytes to determine the type of the message. The access to the buffer is controlled by semaphores. In addition, we have one shared memory segment for each *worker* to send messages to the *boss*. These segments are used in the same manner as the "sending buffer" of the *boss*.

The shared memory version differs from the PVM version in the following respects:

1. The *workers* do not send the pivot column immediately together with the pivot element, i. e., improvement 1. on page 143 is removed: There might be several pivot elements and corresponding columns sent per iteration, depending upon numerical considerations. This behavior could result in an overflow of the shared memory buffer. On the other hand, informing a *worker* to send a column is relatively inexpensive using semaphores.

2. We changed the pivot selection strategy (see 2. on page 143) back to that of the sequential code, mainly because we wanted to have the same pivot selection strategy for an easier comparison of the results and because the additional communication steps are not time-consuming using shared memory and semaphores.

3. We saved some data copies by creating another shared memory segment for the vector z. Thus, in Com(z) the *workers* are notified of the availability of the new vector by a change of the appropriate semaphore value.

Table 7.8 shows the results of the shared memory version on the SUN S20-502.

Example	Sequential		2 processors	
	Time	Iterations	Time	Iterations
`Netlib`	4621.2	130962	5593.3	141486
0321.4	9518.3	21481	7958.2	20465
cre_b	650.5	11121	604.9	13219
nw16	99.6	313	82.2	313
osa030	556.3	2927	545.1	3231
roadnet	801.0	4578	711.2	4644

Table 7.8: Shared memory version on a SUN S20-502

The results on the SUN S20-502 are again not satisfactory. For the `Netlib` problems the times are better than those using PVM, but are still far inferior to the CPLEX sequential times. For the larger models the numbers are even worse. Two contributors to these negative results are the following:

1. The semaphore approach is probably not the right way to exploit shared memory for the fine-grained parallelization necessary in the dual simplex algorithm. It is true that there are other communication primitives available that might be faster. However, as this work was being done, there did not seem to be any better approach available that was portable. We will come to this point again in the next section.

2. There is a serious memory bottleneck in the SUN S20-502 architecture. Because the data bus is rather small, processes running in parallel interfere with each other when accessing memory. Looking at the SPEC results for the single processor (S20-50) and 2-processor (S20-502) models we have

	SUN S20-50	SUN S20-502
SPECrate_int92	1708	3029
SPECrate_fp92	1879	3159

This means that up to about 19% is lost even under ideal circumstances. For memory intensive codes like CPLEX, the numbers are even worse. For the `Netlib` problems, we ran CPLEX alone and twice in parallel on the SUN S20-502:

CPLEX (alone)	CPLEX (twice in parallel)
4621.2 sec.	6584.4 sec.
	6624.7 sec.

This degradation was about 40%! Clearly, the SUN S20-502 has serious limitations in parallel applications[4].

The Silicon Graphics Power Challenge multi-processors are examples of machines that do not suffer from this limitation. Table 7.9 summarizes our tests running the System V semaphore implementation on a two-processor, 75 Mhz Silicon Graphics R8000 multi-processor.

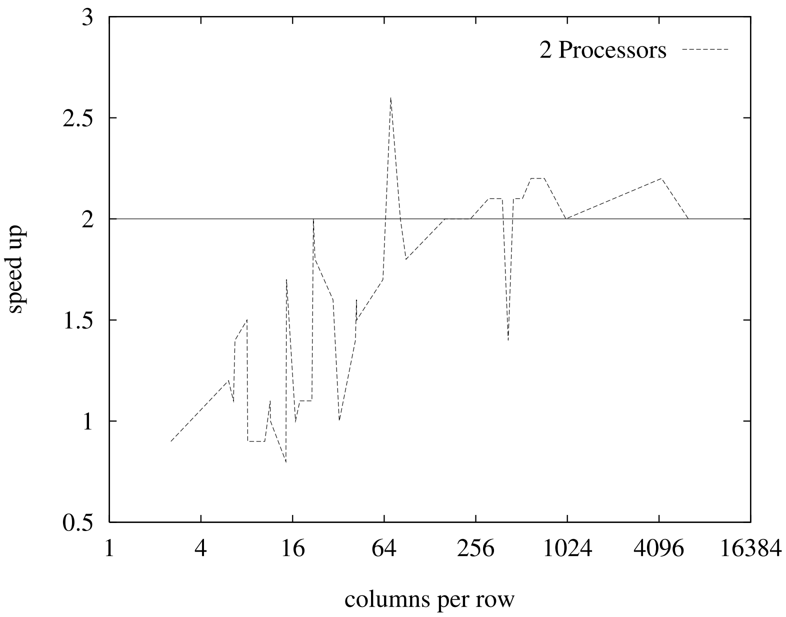


Figure 7.1: Speed up of Shared memory version: All problems

We note that the five larger models (0321.4, cre_b, nw16, osa030, and roadnet) achieve reasonable, though with one exception sublinear speed-ups, ranging from 22% for cre_b to 105% for nw16. One reason that better speed-ups are not obtained is that a significant fraction of the communication costs is independent of problem size – indeed, all steps to the point that the *worker* sends an entering column. As a consequence, examples with low-cost iterations cannot be expected to achieve significant speed-ups. This phenomenon is illustrated by aa25000, sfsu4, nopert, cre_b, mctaq, usfs2, food, aa6, ra1, pilots, and especially the `Netlib` problems (including fit2d), where on the average at most 0.03 seconds are needed per iteration, running sequentially. All other examples, for which the number of iterations of the sequential and parallel codes are roughly equal, give approximately the desired speed-up. The "aa"-examples behave particularly well: The numbers of iterations are constant, individual iterations are expensive, the fraction of work that can be parallelized is near 100%, see Table 7.9.

[4]Sun Microsystems gave us the opportunity to test some of these examples under an optimal environment on their machines. On the SUN S20-502 we got the same results as on our machine, whereas on a SUN S20-712 the degradation was at most 20%. These better results are mainly due to the 1 MB external cache each of the two processors of a SUN S20-712 has. The extra cache helps in avoiding bottlenecks on the data bus.

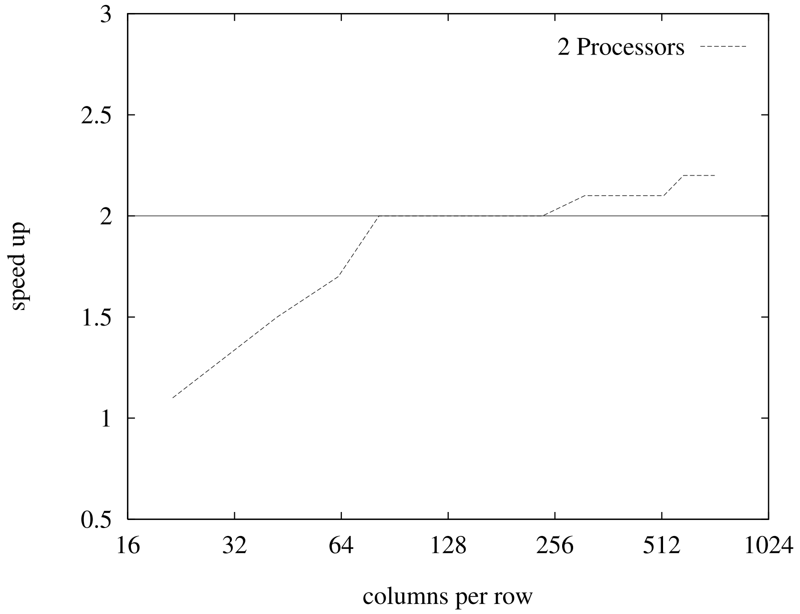


Figure 7.2: Speed up of Shared memory version: “aa”-problems

Finally, note that mctaq, sfsu2, sfsu3, finland, and imp1 fail to follow any particular trend, primarily because the number of iterations for the parallel and sequential codes differ drastically. That such differences arise was unexpected, since the pivot selection strategy in both codes is the same, as is the starting point. However, since the basis is managed by the *boss*, we distribute only the initial non-basic columns among the processors, resulting in a possible column reordering. With this reordering, different columns can be chosen in the Pricing step, leading to different solution paths. Note that in terms of time per iteration, the five listed models do achieve close to linear speed-ups.

Figure 7.1 and 7.2 give a graphical illustration of the numbers in Table 7.9. The abscissa shows the ratio of the number of columns to the number of rows. The ordinate presents the speed-up of all non-`Netlib` examples in Figure 7.1 and all “aa”-examples in Figure 7.2. With the exception of fit2d we obtain at least linear speed-up, when the ratio exceeds 160. For the “aa”-problems we obtain ideal speed-up beginning at a ratio of 80.

7.6 An Implementation Using PowerC

We describe a thread-based parallel implementation of the dual steepest-edge algorithm on an SGI Power Challenge using the SGI PowerC extension of the C programming language.

The work described in this section was carried out at a somewhat later date than that in previous sections. As a result, the base sequential version of CPLEX was somewhat different. As the tables will show, this version not only exhibited improved performance when parallelized, but was significantly faster running sequentially.

In our work we use only a small subset of the compiler directives provided by the PowerC extension: `#pragma parallel`, `#pragma byvalue`, `#pragma local`, `#pragma shared`, `#pragma pfor`, and `#pragma synchronize`. The `parallel` pragma is used to define a parallel *region*. The remaining pragmas are employed inside

Example	Sequential		2 processors		Speed-up
	Time	Iterations	Time	Iterations	
`Netlib`	2004.4	133299	2361.7	138837	0.8
0321.4	4406.2	20677	2681.2	20662	1.6
0341.4	564.8	8225	394.8	8225	1.4
aa100000	257.2	2133	128.8	2133	2.0
aa1000000	15266.6	7902	7030.5	7902	2.2
aa200000	1262.4	4090	632.2	4090	2.0
aa25000	7.9	546	7.1	546	1.1
aa300000	2724.0	5513	1339.5	5513	2.0
aa400000	4068.9	5931	1964.7	5931	2.1
aa50000	34.1	916	23.2	916	1.5
aa500000	6081.8	6747	2878.1	6747	2.1
aa6	22.7	2679	26.2	2679	0.9
aa600000	7619.0	6890	3599.5	6890	2.1
aa700000	9746.5	7440	4536.4	7440	2.1
aa75000	105.1	1419	60.8	1419	1.7
aa800000	11216.1	7456	5172.8	7456	2.2
aa900000	13130.8	7590	6028.9	7590	2.2
amax	3122.5	8276	1923.9	9780	1.6
continent	771.6	16586	558.8	16570	1.4
cre_b	337.8	10654	275.3	10654	1.2
finland	1654.1	24356	1560.7	31416	1.0
fit2d	131.7	6366	97.0	6959	1.4
food	653.5	21433	598.4	21328	1.1
imp1	8252.9	38421	3231.4	30036	2.6
mctaq	531.4	28714	683.1	41460	0.8
nopert	424.1	26648	249.9	24185	1.7
nw16	109.2	403	53.3	403	2.0
osa030	354.8	2943	192.2	2833	1.8
osa060	2182.7	5787	1074.5	5801	2.0
pilots	71.2	4211	82.2	4437	0.9
ra1	51.1	3091	46.2	3091	1.1
roadnet	378.9	4405	213.9	4608	1.8
sfsu2	1818.2	12025	1828.0	23200	1.0
sfsu3	779.2	4055	804.0	9436	1.0
sfsu4	71.5	2256	66.4	2414	1.1
tm	8154.3	74857	5478.7	71657	1.5
us01	782.5	278	350.8	278	2.2
usfs2	241.0	8356	268.5	7614	0.9
w1.dual	27.2	67	13.5	67	2.0

Table 7.9: Shared memory version on a 75 Mhz Silicon Graphics R8000

parallel regions. Their applications and meanings are sketched below.

Defining a parallel region is analogous to defining a C function. The `byvalue`, `local`, and `shared` directives specify the argument list for that function, with each directive specifying the obvious types – for example, `shared` specifies pointers that will be shared by all threads. The `#pragma synchronize` directive forces all threads to complete all computations up to the point of the synchronize statement before any thread is allowed to continue. Exactly one synchronization pragma is used in our implementation (it could be easily avoided by introducing another parallel region). All of the actual parallelism is invoked by the loop-level directive `pfor`.

The key parallel computation is the Pricing step. If this step were carried out in the straightforward way, it's parallelization would also be straightforward, employing the following sort of loop (inside a parallel region):

```
#pragma pfor iterate (j = 0; ncols; 1)
for (j = 0; j < ncols; j++) {
    compute a sparse inner product for column j;
}
```

where `ncols` denotes the number of columns. We note here that the `#pragma pfor` construction means that a parallel region is created for the loop following the pragma, and that within this region the iterates of the loop, the computations of the sparse inner products, will be scheduled at run time on the available processors. For a discussion of the specific scheduling algorithms employed by the compiler see Bauer [1992]. We remark here that on the R8000 the startup cost for the very first parallel region encountered in the code (at run time) is approximately one millisecond; subsequent parallel regions have a startup cost of approximately one microsecond.

Returning to our discussion of Pricing, as noted earlier, CPLEX does not carry out the Pricing step column-wise. In order to exploit sparsity in z (see Step (2)), the part of the constraint matrix corresponding to the non-basic variables at any iteration is stored in a sparse data structure by row, and this data structure is updated at each iteration by deleting the *entering variable* (which is "leaving" the non-basic set) and inserting the *leaving variable*.

Given that A_N is stored by row, the computation of $z^T A_N$ could be parallelized as follows:

```
#pragma pfor iterate (i = 0; nrows; 1)
for (i = 0; i < nrows; i++) {
    α_N += z[i] * (i-th row of A_N);
}
```

where the inner computation itself is a loop computation, and α_N has been previously initialized to 0. The difficulty with this approach is that it creates *false sharing*: the individual entries in α_N will be written to by all threads, causing this data to be constantly moved among the processor caches. One obvious approach to avoiding this difficulty is to create separate target arrays α_{N_p}, one for each thread, with the actual update of α_N carried out as a sequential computation following the computation of the α_{N_p}. However, a much better approach is to directly partition N into subsets, one for each thread. To do so required restructuring a basic CPLEX data structure and the routines that accessed it. Once that was done, the implementation of the parallel pricing was straightforward.

Where K is a multiple of the number of processors, let

$$0 = n_0 \leq n_1 \leq n_2 \leq \ldots \leq n_K = \text{ncols},$$

and let $P_k = \{n_k, \ldots, n_{k+1} - 1\}$ for $k = 0, \ldots, K-1$. The n_k are chosen such that the numbers of non-zeros in A_{P_k} are as nearly equal as possible. For a given set of non-basic indices N, the corresponding partition is then defined by $N_k = N \cap P_k$. Using this partition, the parallel pricing loop takes the form

```
#pragma pfor iterate (k = 0; K; 1)
for (k = 0; k < K; k++) {
   for (i = 0; i < nrows; i++) {
      α_{N_k} += z[i] * (i-th row of A_{N_k});
   }
}
```

In initial testing of the partitioning, an interesting phenomenon was discovered, related at least in part to the cache behavior of the R8000. Consider the model aa400000. Running the sequential code with no partitioning yielded a timing of 2864.1 seconds while the initial PowerC version on two processors using $K = 2$ ran in 1300.4 seconds, a speed-up considerably greater than 2.0. Setting $K = 2$ in the sequential code yielded a run time of 2549.4, much closer to what one would expect. After considerable testing, we thus chose to set K – in both the sequential and parallel instances – to be the smallest multiple of the number of processors that satisfies $K \geq ncols/(50\ nrows)$. Thus, for aa400000 and two processors, K was 8, the smallest multiple of 2 greater than $259924/(50 \cdot 837)$. We note that this change also seems to have benefited other platforms. The dual solution time for fit2d on a 133 Mhz Pentium PC was 204.5 seconds with $K = 1$ and 183.7 with the new setting of $K = 9$.[5] It is very interesting to note that an idea – here the idea of partitioning the column set – that was introduced in the parallel algorithm also improves the sequential code.

We now comment on the remaining steps that were parallelized in the dual algorithm: Enter, Ratio, Pivot, Update-d, and the update of the row-wise representation of A_N.

Ratio and Pivot: For these computations we use the same partition of N used in the Pricing step. Note that the dual algorithm allows the Pricing and Ratio steps to be performed without any intervening computations. As it turned out, in the CPLEX sequential implementation prior to the current work, there were several relatively inexpensive, minor computations that were interspersed between these two major steps. Since entering and leaving parallel regions does incur some fixed costs (see the discussion above), it seemed important to be able to do the Pricing and Ratio steps inside a single region; moreover, with some reorganization within each of these computations, it was possible to carry out the "major part" of each step without introducing synchronization points. Thus, the essential form of the computation as implemented was the following:

```
#pragma pfor iterate (k = 0; K; 1)
for (k = 0; k < K; k++) {
   for (i = 0; i < nrows; i++) {
      α_{N_k} += z[i] * (i-th row of A_{N_k});
   }
   Ratio Test for N_k;
}
```

[5] Dual is not the way to solve fit2d, especially not on a PC. The solution time using simplex primal was 18.6 seconds and using the barrier algorithm 15.4 seconds.

The reorganization of computations for these two steps, as well as other reorganizations to facilitate the parallel computation were carried out so that they also applied when the dual was executed sequentially, thus preserving code unity.

Enter: Since this computation is easy to describe in essentially complete detail, we use it as an illustration of the precise syntax for the PowerC directives:

```
#pragma parallel
#pragma byvalue (nrows)
#pragma local (i_min, min, i)
#pragma shared (x_B, norm, i_min_ar)
{
   i_min = -1;
   min   = 0.0;
   #pragma pfor iterate (i = 0; nrows; 1)
   for (i = 0; i < nrows; i++) {
      if ( x_B[i] < min * norm[i] ) {
         min   = x_B[i] / norm[i];
         i_min = i;
      }
   }
   i_min_ar[mpc_my_threadnum ()] = i_min;
}
i_min = -1;
min   = 0.0;
for (i = 0; i < mpc_numthreads (); i++) {
   if ( i_min_ar[i] != -1 ) {
      if ( x_B[i_min_ar[i]] < min * norm[i_min_ar[i]] ) {
         min   = x_B[i_min_ar[i]] / norm[i_min_ar[i]];
         i_min = i_min_ar[i];
      }
   }
}
```

The PowerC function `mpc_my_threadnum()` returns the index of the thread being executed, an integer from 0 to $T-1$, where T is the total number of threads. The function `mpc_numthreads()` returns T.

A_N **update:** The insertion of new columns is a constant-time operation. However, due to properties of the chosen data structures the deletion operation can be quite expensive. It was parallelized in a straightforward manner.

Finally, we remark on one important computation that was not parallelized. As discussed earlier, the dual steepest-edge algorithms all require the solution of one additional FTRAN per iteration. The result is that two ostensibly "independent" solves are performed using the same basis factorization. These solves are typically quite expensive, and it would seem clear that they should be carried out in parallel (on two processors). However, in the sequential code these two solves have been combined into a single traversal of the factorization structures. That combination, when carefully implemented, results in some reduction in the actual number of computations as well as a very effective use of cache. As a result, all our attempts to separate the computations and perform them in parallel resulted in a degradation in performance.

Computational Results

The computational results for the PowerC parallel dual are given in Table 7.10. Tests were carried out on a 4-processor 75 Mhz R8000. (There was insufficient memory to run aa6000000.)

Comparing the results in Table 7.10 to the profiles in Table 7.1, we see that pilots – as expected, because of the large fraction of intervening non-parallel work – did not achieve ideal performance. On the other hand, cre_b came very close to the ideal speed-up and aa300000 exceeded ideal speed-up by a considerable margin.

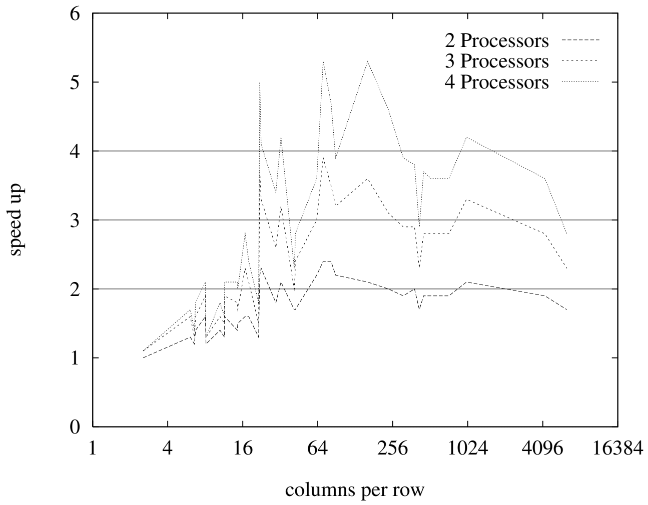


Figure 7.3: Speed-up of PowerC version: All problems

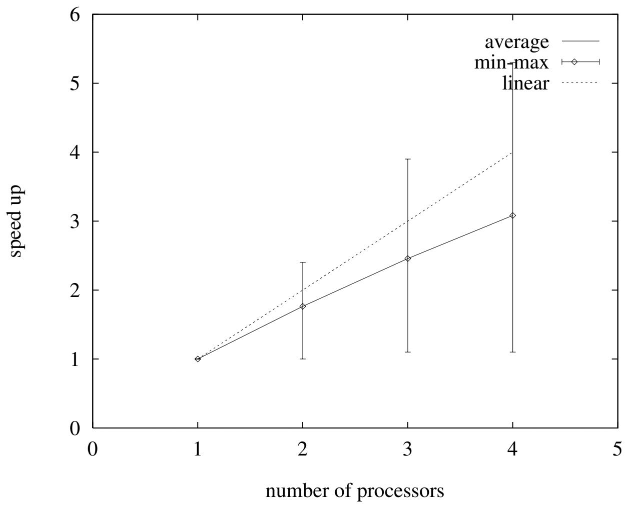


Figure 7.4: Avg. speed-up of PowerC version: All problems

There are unfortunately several, as yet unexplained anomalies in our results. These mainly show up on larger models. In several instances superlinear speed-ups are achieved. Examples are aa200000 and imp1, with 4-processor speed-ups exceeding factors of 5. On the other hand, other models that would seem even more amenable to parallelism, principally the four largest "aa"-models, achieve speed-ups considerably smaller than 4 on 4 processors. At this writing, the authors can offer no better explanation than that these anomalies are due to R8000 cache and memory bus properties.

Figures 7.3 through 7.6 depict the results in Table 7.10 graphically. For 2 processors, linear speed-ups are obtained for all non-`Netlib` problems with ratios 60 or higher. The same is true for 3 processors. An almost ideal speed-up is achieved on 4 processors when the ratio is greater than 70, with the exceptions of fit2d and w1.dual. On 2 processor we see speed-ups of 1.5 for problems with as few as 10 columns per row. Figures 7.4 and 7.6 show almost linear scalability up to 4 processors. In particular, note that the speed-ups for the "aa"-problems are on average close to linear. It remains to determine whether this behavior carries over to more processors.

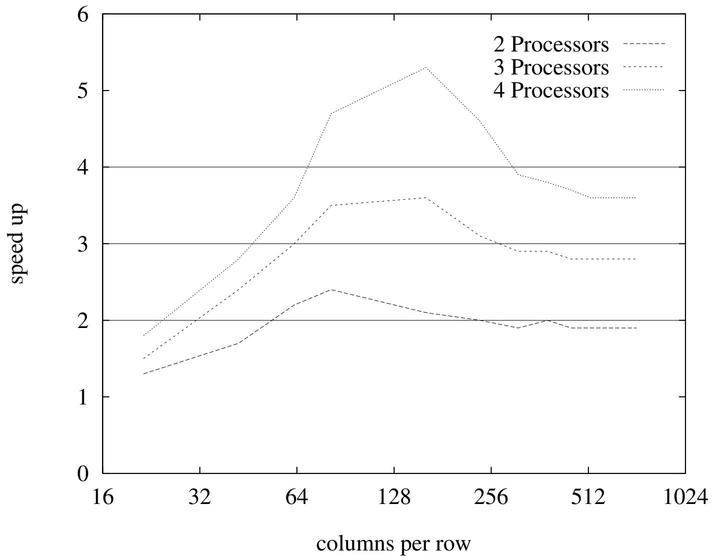


Figure 7.5: Speed-up of PowerC version: "aa"-problems

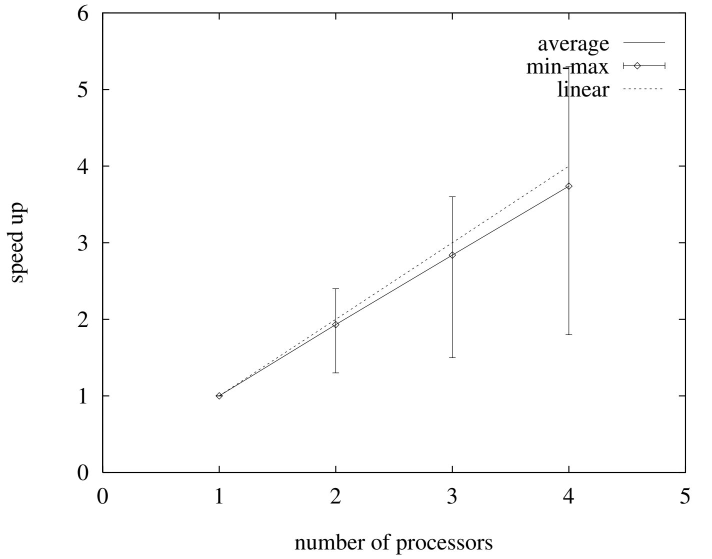


Figure 7.6: Avg. speed-up of PowerC version: "aa"-problems

Example	Iterations	Run time (no. of processors)				Speed-ups		
		1	2	3	4	2	3	4
`Netlib`	136369	1310.2	1216.2	1151.3	1123.6	1.1	1.1	1.2
0321.4	19602	2703.6	1599.4	1218.5	1034.7	1.7	2.2	2.6
0341.4	9190	341.5	205.7	168.3	146.9	1.7	2.0	2.3
aa100000	2280	153.1	64.3	43.3	32.9	2.4	3.5	4.7
aa1000000	7703	7413.5	3851.2	2687.5	2089.4	1.9	2.8	3.6
aa200000	3732	675.4	318.4	189.8	128.1	2.1	3.6	5.3
aa25000	552	3.7	2.9	2.4	2.1	1.3	1.5	1.8
aa300000	5865	1743.1	876.4	557.7	381.7	2.0	3.1	4.6
aa400000	6271	2473.0	1286.5	855.4	629.0	1.9	2.9	3.9
aa50000	1038	20.0	11.5	8.5	7.1	1.7	2.4	2.8
aa500000	6765	3349.5	1713.6	1165.4	879.9	2.0	2.9	3.8
aa6	2509	12.1	10.4	9.5	9.1	1.2	1.3	1.3
aa600000	6668	3904.9	2019.1	1393.0	1054.9	1.9	2.8	3.7
aa700000	7162	4951.4	2542.8	1760.0	1361.5	1.9	2.8	3.6
aa75000	1360	45.8	21.2	15.3	12.6	2.2	3.0	3.6
aa800000	7473	5763.1	3000.6	2084.1	1616.1	1.9	2.8	3.6
aa900000	8166	7242.4	3738.3	2606.4	2020.2	1.9	2.8	3.6
amax	9784	2093.8	1151.7	795.2	625.3	1.8	2.6	3.4
continent	12499	236.7	163.9	141.5	128.9	1.4	1.6	1.8
cre_b	11136	168.5	124.9	107.9	100.5	1.3	1.6	1.7
finland	29497	1086.8	691.4	580.3	526.4	1.6	1.9	2.1
fit2d	5724	49.2	29.3	21.5	17.3	1.7	2.3	2.9
food	21257	311.3	259.5	238.5	223.9	1.2	1.3	1.4
imp1	29297	3424.5	1423.0	868.0	651.5	2.4	3.9	5.3
mctaq	30525	317.0	219.2	177.9	153.0	1.4	1.8	2.1
nopert	27315	197.4	135.9	113.9	99.6	1.5	1.7	2.0
nw16	256	21.9	10.6	6.7	5.2	2.1	3.3	4.2
osa030	2831	154.4	67.8	46.3	37.3	2.3	3.3	4.1
osa060	5753	1197.8	548.1	328.0	241.0	2.2	3.7	5.0
pilots	4196	44.2	42.2	40.9	40.5	1.0	1.1	1.1
ra1	3018	26.4	20.4	17.9	16.6	1.3	1.5	1.6
roadnet	3921	164.5	75.5	51.5	42.3	2.2	3.2	3.9
sfsu2	16286	1724.5	1060.3	761.9	609.3	1.6	2.3	2.8
sfsu3	3692	413.4	201.5	130.8	99.3	2.1	3.2	4.2
sfsu4	3071	56.7	35.8	27.6	23.6	1.6	2.1	2.4
tm	70260	4230.5	2633.7	2232.8	1997.5	1.6	1.9	2.1
us01	245	108.9	57.2	39.2	30.0	1.9	2.8	3.6
usfs2	7962	114.4	83.9	72.2	65.5	1.4	1.6	1.8
w1.dual	67	16.5	9.7	7.2	5.9	1.7	2.3	2.8

Table 7.10: PowerC run times on 1 to 4 processors

Chapter 8

The Intersection of Knapsack Polyhedra

In this chapter we go one step into the direction of exploiting block structure polyhedrally. We try to identify (small) blocks and derive inequalities that simultaneously take all involved constraints of this block into account. The family of inequalities we derive from such a block can be viewed as valid inequalities for the intersection of knapsack polyhedra, where we associate a knapsack polytope with each row of the block. In the context of Part I of this thesis these inequalities are *individual* inequalities, since they are derived from a single block. We relate this family of inequalities to Chvátal-Gomory cutting planes and important special cases such as odd hole and clique inequalities for the stable set polyhedron or families of inequalities for the knapsack polyhedron. We analyze how relations between covering and incomparability numbers associated with the matrix can be used to bound coefficients in these inequalities. For the intersection of several knapsack polyhedra, incomparabilities between the column vectors of the associated matrix will be shown to transfer into inequalities of the associated polyhedron. The inequalities have been incorporated into `SIP` and we report on experimental results.

In detail, this chapter is organized as follows. In Section 8.1 we describe a general family of valid inequalities, called feasible set inequalities, for an integer program that are associated with its feasible solutions. Usually such inequalities must be lifted in order to induce high dimensional faces. We derive lower and upper bounds on the exact lifting coefficients of such an inequality in Section 8.2 and discuss special cases when these bounds can be computed in polynomial time. In Section 8.3 we relate our family of inequalities to Chvátal-Gomory cuts and investigate in Section 8.4 the special case where only two knapsacks are involved in detail. Section 8.5 discusses the separation problem of feasible set inequalities. The use of feasible set inequalities within an implementation for the solution of general mixed integer programming problems is investigated in Section 8.6.

8.1 Feasible Set Inequalities

Consider some finite sets $N, M \subset \mathbb{N}$, some matrix $A \in \mathbb{R}^{M \times N}$, vectors $b \in \mathbb{R}^M, u \in \mathbb{R}^N$, and the polytope

$$P_{\text{IsK}}(N, M, A, b, u) := \operatorname{conv}\{x \in \mathbb{Z}^N : Ax \le b,\ 0 \le x \le u\},$$

that is the convex hull of all integral vectors x satisfying $Ax \le b$ and $0 \le x \le u$. We will assume $N = \{1, \dots, n\}$ and $M = \{1, \dots, m\}$ for $n, m \in \mathbb{N}$ and abbreviate

$P_{\text{IsK}}(N, M, A, b, u)$ by P_{IsK} if it is clear from the context. For convenience, we also use the symbol $P_{\text{IsK}}(S)$ to abbreviate the polytope $P_{\text{IsK}}(S, M, A_{\cdot S}, b, u_S)$.

Definition 8.1.1 *Let $T \subseteq N$ with $\sum_{i\in T} A_{\cdot i}u_i \le b$ and $w : T \mapsto \mathbb{Z}$ be some* weighting *of the elements in T. T is called a* feasible set with respect to the weighting w *if $w^T v \le w^T u_T$ for all $v \in \mathbb{Z}_+^T$ with $v \le u_T$ and $\sum_{i\in T} A_{\cdot i}v_i \le b$. For $j \in N \setminus T$ with $\sum_{i\in T} A_{\cdot i}u_i + A_{\cdot j}u_j \not\le b$, the inequality*

$$\sum_{i\in T} w_i x_i + w_j x_j \le \sum_{i\in T} w_i u_i \tag{8.1}$$

is called a feasible set inequality associated with T (and $\{j\}$) and w *if*

$$w_j \le \min_{l=1,\dots,u_j} \frac{1}{l} \min_x \quad \begin{array}{l} \sum_{i\in T} w_i x_i \\ \sum_{i\in T} A_{\cdot i}x_i \ge A_{\cdot j}\, l - r(T) \\ 0 \le x_i \le u_i, x_i \in \mathbb{Z}, i \in T, \end{array} \tag{8.2}$$

where $r(T) := b - \sum_{i\in T} A_{\cdot i}u_i$.

Theorem 8.1.2 *Feasible set inequalities are valid for $P_{IsK}(T \cup \{j\})$.*

Proof. Let $\gamma = \sum_{i\in T} w_i u_i$ and $\gamma_l := \max\{\sum_{i\in T} w_i x_i : \sum_{i\in T} A_{\cdot i}x_i + A_{\cdot j}l \le b,\ 0 \le x_i \le u_i, x_i \in \mathbb{Z}, i \in T\}$. After complementing variables x_i to $u_i - x_i$ for $i \in T$ we obtain that the right-hand side of (8.2) is $\min_{l=1,\dots,u_j} \frac{1}{l}(\gamma - \gamma_l)$. For some integer solution $\bar{x} \in P_{\text{IsK}}(T \cup \{j\})$ with $\bar{x}_j \ge 1$ we have $\sum_{i\in T} w_i\bar{x}_i + w_j\bar{x}_j \le \gamma_{\bar{x}_j} + \bar{x}_j \min_{l=1,\dots,u_j} \frac{1}{l}(\gamma - \gamma_l) \le \gamma_{\bar{x}_j} + (\gamma - \gamma_{\bar{x}_j}) = \gamma$. In case $\bar{x}_j = 0$ the statement follows from the definition of a feasible set. □

Theorem 8.1.2 states the validity of the feasible set inequality for $P_{\text{IsK}}(T \cup \{j\})$. To obtain a (strong) valid inequality for P_{IsK} we resort to *lifting*, see Appendix C. Consider some permutation $\pi_1, \dots, \pi_{n-|T|-1}$ of the set $N \setminus (T \cup \{j\})$. For $k = 1, \dots, n - |T| - 1$ and $l = 1, \dots, u_{\pi_k}$ let

$$\gamma(k,l) = \max \quad \begin{array}{l} \sum_{i\in T\cup\{j\}} w_i x_i + \sum_{i\in\{\pi_1,\dots,\pi_{k-1}\}} w_i x_i \\ \sum_{i\in T\cup\{j\}} A_{\cdot i}x_i + \sum_{i\in\{\pi_1,\dots,\pi_{k-1}\}} A_{\cdot i}x_i + A_{\cdot \pi_k} l \le b \\ 0 \le x_i \le u_i, x_i \in \mathbb{Z} \text{ for } i \in T \cup \{j, \pi_1, \dots, \pi_{k-1}\}. \end{array} \tag{8.3}$$

Let $\gamma = \sum_{i\in T} w_i u_i$, the lifting coefficients are

$$w_{\pi_k} := \min_{l=1,\dots,u_{\pi_k}} \frac{\gamma - \gamma(k,l)}{l}. \tag{8.4}$$

The following statement is immediate.

Theorem 8.1.3 *The (lifted) feasible set inequality $w^T x \le \sum_{i\in T} w_i u_i$ is valid for P_{IsK}.*

Note that the right-hand side of (8.2) coincides with (8.4) applied to variable j if we substitute in (8.3) the set $T \cup \{j\}$ by T. In other words, a lifted feasible set inequality associated with T and $\{j\}$, where the variables in $N \setminus (T \cup \{j\})$ are lifted according to the sequence $\pi_1, \dots, \pi_{n-|T|-1}$, coincides with the inequality associated with T, where j is lifted first, and the remaining variables $N \setminus (T \cup \{j\})$

are lifted in the same order $\pi_1, \dots, \pi_{n-|T|-1}$. Thus, instead of speaking of a feasible set inequality associated with T and $\{j\}$, we speak in the sequel of a feasible set inequality associated with T and view j as the variable that is lifted first.

Examples of feasible set inequalities include $(1,k)$-configuration, minimal cover, and extended weight inequalities that are known for the knapsack polytope $P_{\mathrm{K}}(N,a,\alpha) = \mathrm{conv}\{x \in \{0,1\}^N : a^T x \leq \alpha\}$ with $a \in \mathbb{R}^N_+, \alpha > 0$, see Section 2.3. Let $S \subseteq N$ be a minimal cover, i.e., $a(S) > \alpha$ and $a(S \setminus \{i\}) \leq \alpha$ for all $i \in S$, and partition S into T and $\{j\}$ for some $j \in S$. Set $w_i := 1$ for all $i \in T$. The feasible set inequality reads $\sum_{i \in T} x_i + w_j x_j \leq |T| = |S| - 1$ with $w_j \leq \min\{|V| : V \subseteq T, \sum_{i \in V} a_i \geq a_j - r(T)\}$. Since $\sum_{i \in T} a_i + r(T) = \alpha$ and $\sum_{i \in S} a_i > \alpha$, this minimum is greater than or equal to one. Therefore, the feasible set inequality is always a $(1,k)$-configuration inequality, see Padberg [1980]. In case the coefficient happens to be one we get a minimal cover inequality. Moreover, feasible set inequalities are just the extended weight inequalities for the knapsack polytope if we choose weights $w_i = 1$ for $i \in T$.

Odd hole- and clique inequalities for the set packing polytope are further examples of lifted feasible set inequalities. For some $0/1$ matrix $A \in \{0,1\}^{M \times N}$, consider the set packing polytope $P_{\mathrm{IsK}}(N, M, A, \mathbb{1}, \mathbb{1}) = \mathrm{conv}\{x \in \{0,1\}^N : Ax \leq \mathbb{1}\}$. Let $G_A = (V, E)$ denote the associated column intersection graph whose nodes correspond to the columns of A and nodes i and j are adjacent if and only if the columns associated with i and j intersect in some row. Let $Q \subseteq V$ be a clique in G_A, then the clique inequality $\sum_{i \in Q} x_i \leq 1$ is valid for P_{IsK}. To see that this inequality is a lifted feasible set inequality, let $T = \{i\}$ for some $i \in Q$. The feasible set inequality $x_i \leq 1$ is valid for $P_{\mathrm{IsK}}(\{i\})$. Lifting the remaining variables $k \in Q \setminus \{i\}$ by applying formula (8.4) yields $w_k = 1$, and the clique inequality follows.

8.2 Bounds on the Lifting Coefficients

For a feasible set inequality associated with T and w_T, the calculation of the lifting coefficients for the variables in $N \setminus T$ requires the solution of an integer program. In this section we study lower and upper bounds for these coefficients. It will turn out that these bounds are sometimes easier to compute. We assume throughout the section that $A \geq 0$ and $w_i \geq 0$ for $i \in T$.

Definition 8.2.1 *Let $T \subseteq N$ be a feasible set with respect to the weighting $w : T \mapsto \mathbb{Z}^T_+$. For $v \in \mathbb{R}^m$ we define the*

Covering Number

$$\phi^{\geq}(v) := \min\{\sum_{i \in T} w_i x_i : \sum_{i \in T} A_{\cdot i} x_i \geq v,\ 0 \leq x_i \leq u_i, x_i \in \mathbb{Z}, i \in T\},$$

$\geq$-Incomparability Number

$$\begin{aligned}\phi^{\ngeq}(v) := \min\{\sum_{i \in T} w_i x_i : \ & \sum_{i \in T} A_{\cdot i} x_i \ngeq v,\ 0 \leq x_i \leq u_i, x_i \in \mathbb{Z}, i \in T, \\ & \wedge \exists\, j \in T, x_j < u_j : \sum_{i \in T} A_{\cdot i} x_i + A_{\cdot j} \geq v\},\end{aligned}$$

$\leq$-Incomparability Number

$$\phi^{\nleq}(v) := \min\{\sum_{i \in T} w_i x_i : \sum_{i \in T} A_{\cdot i} x_i \nleq v,\ 0 \leq x_i \leq u_i, x_i \in \mathbb{Z}, i \in T\},$$

where we set $\phi^{\ngeq}(v) := 0$ and $\phi^{\nleq}(v) := 0$ for $v \leq 0$.

Note that the covering number $\phi^{\geq}(\cdot)$ is a generalization of the covering number $\phi(\cdot)$ defined in (2.6) for the 0/1 knapsack problem. Consider a (lifted) feasible set inequality $w^T x \leq \sum_{i\in T} w_i u_i$ associated with T and w_T, where the variables in $N \setminus T$ are lifted in the sequence $\pi_1, \dots, \pi_{n-|T|}$. The following proposition gives upper bounds for the lifting coefficients derived from the covering number.

Proposition 8.2.2

(a) $w_{\pi_1} = \min_{l=1,\dots,u_{\pi_1}} \frac{1}{l}\, \phi^{\geq}(A_{\cdot\pi_1} l - r(T))$.

(b) $w_{\pi_k} \leq \min_{l=1,\dots,u_{\pi_k}} \frac{1}{l}\, \phi^{\geq}(A_{\cdot\pi_k} l - r(T))$, *for* $k = 2, \dots, n-|T|$.

Proof. (a) directly follows from Theorem 8.1.2. To see (b), it suffices to show that $\gamma - \gamma(k,l) \leq \phi^{\geq}(A_{\cdot\pi_k} l - r(T))$ for $l = 1, \dots, u_{\pi_k}$, see (8.4). This relation is obtained by

$$
\begin{array}{rcl}
\gamma - \gamma(k,l) = & \gamma - \max & \sum_{i\in T} w_i x_i + \sum_{i\in\{\pi_1,\dots,\pi_{k-1}\}} w_i x_i \\
 & & \sum_{i\in T} A_{\cdot i} x_i + \sum_{i\in\{\pi_1,\dots,\pi_{k-1}\}} A_{\cdot i} x_i + A_{\cdot\pi_k} l \leq \gamma, \\
 & & 0 \leq x \leq u_{T\cup\{\pi_1,\dots,\pi_{k-1}\}}, x \in \mathbb{Z}^{T\cup\{\pi_1,\dots,\pi_{k-1}\}} \\
= & \min & \sum_{i\in T} w_i x_i - \sum_{i\in\{\pi_1,\dots,\pi_{k-1}\}} w_i x_i \\
 & & \sum_{i\in T} A_{\cdot i} x_i - \sum_{i\in\{\pi_1,\dots,\pi_{k-1}\}} A_{\cdot i} x_i \geq A_{\cdot\pi_k} l - r(T), \\
 & & 0 \leq x \leq u_{T\cup\{\pi_1,\dots,\pi_{k-1}\}}, x \in \mathbb{Z}^{T\cup\{\pi_1,\dots,\pi_{k-1}\}} \\
\leq & \min & \sum_{i\in T} w_i x_i \\
 & & \sum_{i\in T} A_{\cdot i} x_i \geq A_{\cdot\pi_k} l - r(T), \\
 & & 0 \leq x \leq u_T, x \in \mathbb{Z}^T \\
= & \phi^{\geq}(A_{\cdot\pi_k} l - r(T)), &
\end{array}
$$

where the second equation follows by complementing variables x_i, $i \in T$. $\square$

To derive lower bounds on the lifting coefficients we need the following relations.

Lemma 8.2.3 *For* $v_1, v_2 \in \mathbb{R}^m$ *with* $v_1, v_2 \geq 0$ *holds:*

(a) $\phi^{\geq}(v_1) \geq \phi^{\not\geq}(v_1)$ *and* $\phi^{\geq}(v_1) \geq \phi^{\not\leq}(v_1)$.

(b) $\phi^{\geq}, \phi^{\not\geq}$, *and* $\phi^{\not\leq}$ *are monotonically increasing, that is for* $v_1 \geq v_2$, $\phi^{\geq}(v_1) \geq \phi^{\geq}(v_2)$, $\phi^{\not\geq}(v_1) \geq \phi^{\not\geq}(v_2)$, *and* $\phi^{\not\leq}(v_1) \geq \phi^{\not\leq}(v_2)$.

(c) $\phi^{\geq}(v_1 + v_2) \geq \phi^{\not\geq}(v_1) + \phi^{\not\leq}(v_2)$.

(d) $\phi^{\geq}(v_1 + v_2) + \max\{w_i : i \in T\} \geq \phi^{\not\leq}(v_1) + \phi^{\geq}(v_2)$.

(e) $\phi^{\not\leq}(v_1 + v_2) + \max\{w_i : i \in T\} \geq \phi^{\not\leq}(v_1) + \phi^{\not\leq}(v_2)$.

Proof. (a) and (b) are obvious, the proofs of (c) and (e) follow the same line. We show exemplarily (c). Let $\bar{x} \in \mathbb{R}^T$ with $0 \leq \bar{x} \leq u_T$ and $\sum_{i\in T} A_{\cdot i}\bar{x}_i \geq v_1 + v_2$ such that $\sum_{i\in T} w_i \bar{x}_i = \phi^{\geq}(v_1 + v_2)$. If $v_1 = v_2 = 0$ the statement is trivial. Otherwise, suppose w.l.o.g. $v_1 > 0$. Let $z \in \mathbb{R}^T$ with $z \leq \bar{x}$ such that $\sum_{i\in T} A_{\cdot i} z_i \geq v_1$ and $\sum_{i\in T} w_i z_i$ is minimal. Since $v_1 > 0$ there exists some $i_0 \in T$ with $z_{i_0} > 0$, and since z was chosen to be minimal, we have that $\sum_{i\in T} A_{\cdot i} z_i - A_{\cdot i_0} \not\geq v_1$. This implies that $\sum_{i\in T} A_{\cdot i}(\bar{x}_i - z_i) + A_{\cdot i_0} \not\leq v_2$. Summing up, we get $\phi^{\geq}(v_1 + v_2) = \sum_{i\in T} w_i \bar{x}_i = (\sum_{i\in T} A_{\cdot i} z_i - A_{\cdot i_0}) + (\sum_{i\in T} A_{\cdot i}(\bar{x}_i - z_i) + A_{\cdot i_0}) \geq \phi^{\not\geq}(v_1) + \phi^{\not\leq}(v_2)$. Finally, (d) directly follows from (c) and the fact that $\phi^{\not\geq}(v_1) + \max\{w_i : i \in T\} \geq \phi^{\geq}(v_1)$. $\square$

With the help of Lemma 8.2.3 we are able to bound the lifting coefficients from below.

Theorem 8.2.4 *For $k = 1, \dots, n - |T|$ we have*

$$w_{\pi_k} \geq \min_{l=1,\dots,u_{\pi_k}} \frac{\phi^{\not\leq}(A_{\cdot\pi_k} l - r(T))}{l} - \max\{w_i : i \in T\}. \tag{8.5}$$

Proof. Let $c_{\pi_k} := \min_{l=1,\dots,u_{\pi_k}} \frac{\phi^{\not\leq}(A_{\cdot\pi_k} l - r(T))}{l} - \max\{w_i : i \in T\}$ denote the right-hand side of (8.5), for $k = 1, \dots, n - |T|$. We show by induction on k that the inequality $\sum_{i \in T} w_i x_i + \sum_{i=1}^{k} c_{\pi_i} x_i \leq \sum_{i \in T} w_i u_i$ is valid. For $k = 1$, the statement follows from Proposition 8.2.2 (a) and Lemma 8.2.3 (a). Now let $k \geq 2$ and suppose the statement is true for all $l < k$. Let $\bar{x}$ be an optimal solution of

$$\begin{aligned} \max \quad & \sum_{i \in T} w_i x_i + \sum_{i=1}^{k} c_{\pi_i} x_{\pi_i} \\ & \sum_{i \in T} A_{\cdot i} x_i + \sum_{i=1}^{k} A_{\cdot \pi_i} x_{\pi_i} \leq b \\ & 0 \leq x_i \leq u_i, x_i \in \mathbb{Z} \text{ for } i \in T \cup \{\pi_1, \dots, \pi_k\}. \end{aligned}$$

We must show that $\sum_{i \in T} w_i \bar{x}_i + \sum_{i=1}^{k} c_{\pi_i} \bar{x}_{\pi_i} \leq \sum_{i \in T} w_i u_i$. First note that the inequality is valid if $\bar{x}_{\pi_k} = 0$. This is equivalent to saying

$$\phi^{\geq}(\sum_{i=1}^{k-1} A_{\cdot\pi_i} x_{\pi_i} - r(T)) \geq \sum_{i=1}^{k-1} c_{\pi_i} x_{\pi_i},$$

for all $x \in \mathbb{Z}^{\{\pi_1,\dots,\pi_{k-1}\}}$ with $\sum_{i=1}^{k-1} A_{\cdot\pi_i} x_{\pi_i} \leq b$, $0 \leq x_{\pi_i} \leq u_{\pi_i}$, $i = 1, \dots, k-1$. Applying Lemma 8.2.3 (d) and (b) we obtain with $w_{max} := \max\{w_i : i \in T\}$

$$\begin{aligned} \phi^{\geq}\Big(\sum_{i=1}^{k} A_{\cdot\pi_i} \bar{x}_{\pi_i} - r(T)\Big) &\geq \phi^{\geq}\Big(\sum_{i=1}^{k-1} \big(A_{\cdot\pi_i} \bar{x}_{\pi_i} - r(T)\big)\Big) + \phi^{\not\leq}(A_{\cdot\pi_k} \bar{x}_{\pi_k}) - w_{max} \\ &\geq \sum_{i=1}^{k-1} c_{\pi_i} \bar{x}_{\pi_i} + \phi^{\not\leq}(A_{\cdot\pi_k} \bar{x}_{\pi_k} - r(T)) - w_{max} \\ &\geq \sum_{i=1}^{k-1} c_{\pi_i} \bar{x}_{\pi_i} + \bar{x}_{\pi_k} \min_{l=1,\dots,u_k} \frac{\phi^{\not\leq}(A_{\cdot\pi_k} l - r(T))}{l} - w_{max} \\ &= \sum_{i=1}^{k} c_{\pi_i} \bar{x}_{\pi_i}. \end{aligned}$$

On account of $\sum_{i \in T} w_i (u_i - \bar{x}_i) \geq \phi^{\geq}(\sum_{i=1}^{k} A_{\cdot\pi_i} \bar{x}_{\pi_i} - r(T))$ the statement follows. □

As a corollary of Proposition 8.2.2 and Theorem 8.2.4 we obtain the relation (2.8) of extended weight inequalities. Theorem 8.2.4 applies, in particular, if we set the coefficient of the first lifted variable w_{π_1} to the upper bound of Proposition 8.2.2 (a). The subsequent example shows that in this case the lower bounds given in Theorem 8.2.4 may be tight.

Example 8.2.5 *Let* $A = \begin{bmatrix} 1 & 1 & 6 & 6 & 8 & 6 & 1 & 3 \\ 4 & 6 & 1 & 1 & 3 & 9 & 2 & 1 \end{bmatrix}$ *and* $b = \begin{bmatrix} 14 \\ 12 \end{bmatrix}$. *The set* $T = \{1, 2, 3, 4\}$ *with weights* $w_i = 1$, $i \in T$ *is feasible for the* $0/1$ *program* $\max\{c^T x : x \in P\}$ *with* $P := \operatorname{conv}\{x \in \{0,1\}^8 : Ax \leq b\}$. *We obtain* $\phi^{\geq}\binom{8}{3} = 3$ *and* $\phi^{\not\leq}\binom{6}{9} = 2$, *because* $\binom{6}{9} \geq A_{\cdot i}$ *for* $i \in T$. *Accordingly we get* $\phi^{\not\leq}\binom{1}{2} = 1 = \phi^{\not\leq}\binom{3}{1}$. *The inequality* $x_1 + x_2 + x_3 + x_4 + \phi^{\geq}\binom{8}{3} x_5 + \sum_{i=6}^{8} (\phi^{\not\leq}(A_{\cdot i}) - 1) x_i \leq 4$ *reads* $x_1 + x_2 + x_3 + x_4 + 3x_5 + x_6 \leq 4$. *It defines a facet of* P.

The question remains, whether the values $\phi^{\geq}, \phi^{\ngeq}$ and $\phi^{\nleq}$ are easier to compute than the exact lifting coefficient. Indeed, they sometimes are. Suppose $w_i = 1$ for all $i \in T$ and consider the *comparability digraph* $G = (V, E)$ that is obtained by introducing a node for each column and arcs (i, j) if $A_{\cdot i} \geq A_{\cdot j}$ and $A_{\cdot i} \neq A_{\cdot j}$ or if $A_{\cdot i} = A_{\cdot j}$ and $i > j$, for $i, j \in \{1, \ldots, n\}$ (where transitive arcs may be deleted). Let r denote the number of nodes with indegree zero, i.e., $|\delta^-(i)| = 0$. Then, $\phi^{\geq}, \phi^{\ngeq}$ and $\phi^{\nleq}$ can be computed in time $O(n^r + \alpha)$, where α is the time to construct the comparability digraph. For example, in case of one knapsack inequality the comparability digraph turns out to be a path, and thus $\phi^{\geq}, \phi^{\ngeq}$ and $\phi^{\nleq}$ can be computed in time $O(n + n \log n) = O(n \log n)$. Compare hereto the discussions in Section 5.3.

8.3 Connection to Chvátal-Gomory Cuts

So far we have been discussing feasible set inequalities for general integer programs. Since we have not subsumed any assumptions on the underlying constraint matrix A, a comparison to Chvátal-Gomory cutting planes that do not rely on any particular structure of A is natural. Recall that Chvátal-Gomory inequalities for the system $Ax \leq b$, $0 \leq x \leq u$, $x \in \mathbb{Z}^n$ are cutting planes $d^T x \leq \delta$ such that $d_i = \lfloor \lambda^T \hat{A}_{\cdot i} \rfloor$, $i = 1, \ldots, n$, and $\delta = \lfloor \lambda^T \hat{b} \rfloor$ for some $\lambda \in \mathbb{R}_+^{m+n}$, where $\hat{A} = \begin{bmatrix} A \\ I \end{bmatrix}$ and $\hat{b} = \begin{bmatrix} b \\ u \end{bmatrix}$.

Consider a (lifted) feasible set inequality $w^T x \leq \sum_{i \in T} w_i u_i$ associated with T and w_T, whose remaining variables $N \setminus T$ are lifted in the sequence $\pi_1, \ldots, \pi_{n-|T|}$. This lifted feasible set inequality is compared to Chvátal-Gomory inequalities resulting from multipliers $\lambda \in \mathbb{R}_+^{m+n}$ that satisfy $\lfloor \lambda^T \hat{A}_{\cdot i} \rfloor = w_i$ for $i \in T$.

Proposition 8.3.1

(a) $\lfloor \lambda^T \hat{b} \rfloor \geq \sum_{i \in T} u_i w_i$.

(b) If $\lfloor \lambda^T \hat{b} \rfloor = \sum_{i \in T} u_i w_i$, *let* j *be the smallest index with* $\lfloor \lambda^T \hat{A}_{\cdot \pi_j} \rfloor \neq w_{\pi_j}$. *Then,* $\lfloor \lambda^T \hat{A}_{\cdot \pi_j} \rfloor < w_{\pi_j}$.

Proof. Since T is a feasible set, (a) is obviously true. To see (b) suppose the contrary and let j be the first index with $\lfloor \lambda^T \hat{A}_{\cdot \pi_j} \rfloor \neq w_{\pi_j}$ and $\lfloor \lambda^T \hat{A}_{\cdot \pi_j} \rfloor > w_{\pi_j}$. Set $\gamma = \sum_{i \in T} u_i w_i$ and consider an optimal solution $\bar{x} \in \mathbb{Z}^{T \cup \{\pi_1, \ldots, \pi_j\}}$ of (8.3) such that $w_{\pi_j} = \frac{\gamma - \gamma(j, \bar{x}_j)}{\bar{x}_j}$. Obviously, $\bar{x}$ can be extended to a feasible solution $\tilde{x}$ of P_{IsK} by setting $\tilde{x}_i = \bar{x}_i$, if $i \in T \cup \{\pi_1, \ldots, \pi_j\}$, $\tilde{x}_i = 0$, otherwise. This solution satisfies the feasible set inequality with equality, since $w^T \tilde{x} = \sum_{i \in T \cup \{\pi_1, \ldots, \pi_{j-1}\}} w_i \tilde{x}_i + w_{\pi_j} \tilde{x}_j = \gamma(j, \tilde{x}_j) + \frac{\gamma - \gamma(j, \tilde{x}_j)}{\tilde{x}_j} \tilde{x}_j = \gamma$. On the other hand, j is the first index where the Chvátal-Gomory and the feasible set coefficient differ and we conclude $\sum_{i \in N} \lfloor \lambda^T \hat{A}_{\cdot i} \rfloor \tilde{x}_i = \sum_{i \in T \cup \{\pi_1, \ldots, \pi_{j-1}\}} \lfloor \lambda^T \hat{A}_{\cdot i} \rfloor \tilde{x}_i + \lfloor \lambda^T \hat{A}_{\cdot \pi_j} \rfloor \tilde{x}_j > \sum_{i \in T \cup \{\pi_1, \ldots, \pi_{j-1}\}} w_i \tilde{x}_i + w_{\pi_j} \tilde{x}_j = \gamma = \lfloor \lambda^T \hat{b} \rfloor$, contradicting the validity of the Chvátal-Gomory inequality. $\square$

As soon as the first two coefficients differ, for $k \in \{\pi_{j+1}, \ldots, \pi_{n-|T|}\}$, no further statements on the relations of the coefficients are possible, in general.

Example 8.3.2 *For some number* $b \in \mathbb{N}$, *consider the knapsack polytope* $P(b) = P_K(\{1, \ldots, 7\}, (2, 6, 8, 9, 9, 21, 4)^T, b)$, *i.e., the convex hull of all* $0/1$ *solutions that satisfy the knapsack inequality*

$$2x_1 + 6x_2 + 8x_3 + 9x_4 + 9x_5 + 21x_6 + 4x_7 \leq b.$$

One Chvátal-Gomory cutting plane for $P(b)$ reads

$$x_1 + 3x_2 + 4x_3 + 4x_4 + 4x_5 + 10x_6 + 2x_7 \le \lfloor b/2 \rfloor.$$

Let $b = 25$. The set $T := \{1,2,3,4\}$ is a feasible set with respect to the weights $w_1 = 1$, $w_2 = 3$, $w_3 = w_4 = 4$. Lifting the items $5,6,7$ in this order we obtain the lifted feasible set inequality that is valid for $P(25)$:

$$x_1 + 3x_2 + 4x_3 + 4x_4 + 4x_5 + 11x_6 + x_7 \le 12.$$

The right-hand side of the Chvátal-Gomory cutting plane and the lifted feasible set inequality coincide. With respect to the lifting order $5,6,7$ the coefficient of item 6 is the first one in which the two inequalities differ. This coefficient is 11 in the feasible set inequality and 10 in the Chvátal-Gomory cutting plane. For item 7 the coefficient in the feasible set inequality is then smaller than the corresponding one in the Chvátal-Gomory cutting plane. For $b = 27$ we obtain the lifted feasible set inequality that is valid for $P(27)$:

$$x_1 + 3x_2 + 4x_3 + 4x_4 + 4x_5 + 9x_6 + x_7 \le 12.$$

The right-hand side of this inequality is by one smaller than the right-hand side of the corresponding Chvátal-Gomory cutting plane for $\lambda = \frac{1}{2}$. However, the coefficients of the items 6 and 7 are smaller than the corresponding coefficients of the Chvátal-Gomory cutting plane.

Under certain conditions a feasible set- and a Chvátal-Gomory cutting plane coincide.

Theorem 8.3.3 *Consider the integer program* $\max\{c^T x : Ax \le b,\ 0 \le x \le \mathbb{1},\ x \in \mathbb{Z}^n\}$ *with $A \in \mathbb{N}^{m \times n}$. Let T be a feasible set with respect to the weights $\lfloor \lambda^T \hat{A}_{\cdot i} \rfloor = w_i$, $i \in T$, for some $\lambda \in \mathbb{R}^m_+$. Further assume $\lfloor \lambda^T b \rfloor = \sum_{i \in T} \lfloor \lambda^T A_{\cdot i} \rfloor$. If, for all $j \in N \setminus T$, column vector $A_{\cdot j}$ is a $0/1$-combination of elements from $\{A_{\cdot i} : i \in T\}$, then*

$$\lfloor \lambda^T A_{\cdot j} \rfloor = \phi^{\ge}(A_{\cdot j} - r(T)).$$

Proof. We first show that $\lfloor \lambda^T A_{\cdot j} \rfloor \ge \phi^{\ge}(A_{\cdot j} - r(T))$. Since $A_{\cdot j}$ is a $0/1$-combination of elements from $\{A_{\cdot i} : i \in T\}$, there exist $\sigma \in \{0,1\}^T$ such that $\sum_{i \in T} \sigma_i A_{\cdot i} = A_{\cdot j}$. Thus, $\lfloor \lambda^T A_{\cdot j} \rfloor = \lfloor \lambda^T (\sum_{i \in T} \sigma_i A_{\cdot i}) \rfloor \ge \sum_{i \in T} \sigma_i \lfloor \lambda^T A_{\cdot i} \rfloor = \sum_{i \in T} \sigma_i w_i \ge \phi^{\ge}(A_{\cdot j} - r(T))$.

To show the opposite relation, we know by Proposition 8.2.2 that the coefficient of the feasible set inequality w_j satisfies $w_j \le \phi^{\ge}(A_{\cdot j} - r(T))$. By Proposition 8.3.1 (b), however, we know that, if w_j does not coincide with $\lfloor \lambda^T A_{\cdot j} \rfloor$ for all $j \in N \setminus T$, there is at least one index j with $w_j > \lfloor \lambda^T A_{\cdot j} \rfloor$. This together with the first part of the proof implies $\phi^{\ge}(A_{\cdot j} - r(T)) \le \lfloor \lambda^T A_{\cdot j} \rfloor < w_j \le \phi^{\ge}(A_{\cdot j} - r(T))$, a contradiction. Thus, $w_j = \lfloor \lambda^T A_{\cdot j} \rfloor$ for all j, and the claim follows by Proposition 8.2.2. □

By Proposition 8.2.2 the expression $\phi^{\ge}(A_{\cdot j} - r(T))$ is an upper bound on the exact coefficient of an item $j \in N \setminus T$ in any lifted feasible set inequality associated with T and the weighting $\lfloor \lambda^T A_{\cdot i} \rfloor$, $i \in T$. On the other hand, the Chvátal-Gomory cutting plane $\sum_{j \in N} \lfloor \lambda^T A_{\cdot j} \rfloor \le \lfloor \lambda^T b \rfloor$ is valid for P_{IsK}. Therefore, this Chvátal-Gomory cutting plane must coincide with any lifted feasible set inequality $\sum_{i \in T} \lfloor \lambda^T A_{\cdot i} \rfloor x_i + \sum_{j \in N \setminus T} w_j x_j \le \lfloor \lambda^T b \rfloor$ independent on the sequence in which the lifting coefficients w_j for the items in $N \setminus T$ are computed.

8.4 Consecutively Intersecting Knapsacks

So far we have been discussing a framework with which one can define and explain families of cutting planes for a general integer program. On the other hand, from a practical point of view the cutting plane phase of usual codes for integer programming relies in particular on valid inequalities for knapsack polyhedra. From both a theoretical and a practical point of view it would be desirable to understand under what conditions facets of single knapsack polyhedra define or do not define strong cutting planes of an integer program when several knapsack constraints intersect. This question is addressed in this section. In fact, here we study a special family of 0/1 programs that arises when $A \in \mathbb{N}^{m \times n}$ and $Ax \leq b$ defines a system of *consecutively intersecting knapsack constraints.* Throughout this section we assume $u = \mathbb{1}$, $A \geq 0$ and integral. For $i = 1, \ldots, m$, let $N_i := \operatorname{supp}(A_{i\cdot})$ and $P^i := P_{\mathrm{K}}(N_i, A_{i\cdot}, b_i) = \operatorname{conv}\{x \in \{0,1\}^{N_i} : \sum_{j \in N_i} a_{ij}x_j \leq b_i\}$.

Definition 8.4.1 *A system of linear inequalities $Ax \leq b$ is called a system of* consecutively intersecting knapsack constraints *if $A \in \mathbb{N}^{m \times n}$ and $N_i \cap N_l = \emptyset$ for all $i, l \in \{1, \ldots, m\}$, $|i - l| \geq 2$.*

A natural question when one starts investigating the intersection of several knapsack polyhedra is when this polyhedron inherits all the facets of the single knapsack polyhedra.

Proposition 8.4.2 *Let $A \in \mathbb{N}^{m \times n}$ and $Ax \leq b$ be a system of consecutively intersecting knapsack constraints. Let $i \in \{1, \ldots, m\}$. If $N_i \cap N_l \cup \{k\}$ is not a cover for all $l \in \{i-1, i+1\}$ and $k \notin N_i$, then every facet-defining inequality of the knapsack polyhedron P^i defines a facet of P_{IsK}.*

Proof. Suppose that $\sum_{j \in N_i} c_j x_j \leq \gamma$ defines a facet of P^i. Let $\sum_{j \in N} d_j x_j \leq \gamma$ define a facet F of P_{IsK} such that

$$F_c := \{x \in P_{\mathrm{IsK}} : \sum_{j \in N_i} c_j x_j = \gamma\} \subseteq F.$$

Let $x^\circ \in \mathbb{Z}^n \cap F_c$ be a vector such that $x_j^\circ = 0$ for all $j \notin N_i$.

Consider some $j \notin N_i$, and let $j \in N_l$ for some $l \neq i$. If $|l - i| \geq 2$, obviously $x^\circ + e_j \in F_c$. In the other case, we know $N_i \cap N_l \cup \{j\}$ is not a cover (on account of the conditions in the proposition), and thus $x^\circ + e_j \in F_c$ as well. This implies that $d_j = 0$. Because $\sum_{j \in N_i} c_j x_j \leq \gamma$ defines a facet of P^i we obtain

$$\dim(F_c) \geq \dim(P^i) - 1 + |N| - |N_i| = |N| - 1 \geq \dim(P_{\mathrm{IsK}}) - 1.$$

Therefore, F_c defines a facet of P_{IsK} that coincides with F. $\square$

The condition that, for every $k \notin N_i$, $l = i-1, i+1$, the set $(N_i \cap N_l) \cup \{k\}$ is not a cover, is essential for the correctness of the proposition as the following example shows.

Example 8.4.3 *For $b \in \mathbb{N} \setminus \{0\}$ let A be the matrix* $\begin{bmatrix} \frac{b}{3}+1 & \frac{b}{3}+1 & \frac{b}{3}+1 & 0 \\ 0 & \frac{b}{3}+1 & \frac{b}{3}+1 & b \end{bmatrix}$ *and consider $P_{IsK} = \operatorname{conv}\{x \in \{0,1\}^4 : Ax \leq b\}$. Then $N_1 \cap N_2 = \{2,3\}$, and the set $\{2,3,4\}$ defines a cover.*

The inequality $x_1 + x_2 + x_3 \leq 2$ defines a facet of the knapsack polyhedron

$$\operatorname{conv}\{x \in \{0,1\}^3 : (\frac{b}{3}+1)x_1 + (\frac{b}{3}+1)x_2 + (\frac{b}{3}+1)x_3 \leq b\}.$$

On the other hand, the inequality $x_1 + x_2 + x_3 \leq 2$ is not facet-defining for P_{IsK}, since the face $F = \{x \in P_{IsK} : x_1 + x_2 + x_3 \leq 2\}$ is strictly contained in the face induced by the inequality $x_1 + x_2 + x_3 + x_4 \leq 2$.

In certain cases, a complete description of P_{IsK} may even be derived from the description of the single knapsack polyhedra.

Theorem 8.4.4 *Let $m = 2$ and $A \in \mathbb{N}^{2\times n}$. Let $Ax \le b$ be a system of consecutively intersecting knapsack constraints such that every pair of items from $N_1 \cap N_2$ is a cover. For $i = 1, 2$ let $C^i x \le \gamma^i$ be a system of inequalities that describes the single knapsack polyhedron P^i.*

Then, P_{IsK} is described by the system of inequalities

$$\begin{array}{lcl} \sum_{j\in N_1\cap N_2} x_j & \le & 1 \\ C^i x & \le & \gamma^i \textit{ for } i = 1, 2. \end{array}$$

Proof. Note that the system of linear inequalities given in the theorem is valid for P_{IsK}. To see that it suffices to describe P_{IsK}, let $c^T x \le \gamma$ be a non-trivial facet-defining inequality of P_{IsK} that is not a positive multiple of the inequality $\sum_{j\in N_1\cap N_2} x_j \le 1$.W.l.o.g. we assume that $\text{supp}(c) \cap N_1 \ne \emptyset$. Let $Z = \{z_1, \ldots, z_t\} = N_1 \cap N_2$. We claim that $(\text{supp}(c) \setminus Z) \cap N_2 = \emptyset$. Suppose the contrary. We define natural numbers $\gamma^0, \gamma^1, \ldots, \gamma^t$ by

$$\begin{aligned} \gamma^0 &:= \max\{\sum_{j\in N_1\setminus Z} c_j x_j : \sum_{j\in N_1\setminus Z} a_{1j} x_j \le b_1,\ x \in \{0,1\}^{N_1\setminus Z}\}, \\ \gamma^i &:= \max\{\sum_{j\in N_1\setminus Z} c_j x_j : \sum_{j\in N_1\setminus Z} a_{1j} x_j \le b_1 - a_{1z_i},\ x \in \{0,1\}^{N_1\setminus Z}\}, \end{aligned}$$

and claim that the face induced by the inequality $c^T x \le \gamma$ is contained in the face induced by the inequality

$$\sum_{j\in N_1\setminus Z} c_j x_j + \sum_{i=1}^{t} (\gamma^0 - \gamma^i) x_{z_i} \le \gamma^0. \tag{8.6}$$

This inequality is valid for P_{IsK} by definition of the values γ^i, $i = 0, \ldots, t$ and because $\sum_{i\in Z} x_i \le 1$.

Let $x \in P_{\text{IsK}} \cap \mathbb{Z}^N$ such that $c^T x = \gamma$. If $\sum_{i\in Z} x_i = 0$, then $c^T x = \gamma^0$, since otherwise we obtain a contradiction to the validity of $c^T x \le \gamma$. If $\sum_{i\in Z} x_i > 0$, then $\sum_{i\in Z} x_i = 1$. Let $z_i \in Z$ with $x_{z_i} = 1$. Then $c^T x = \gamma$ implies that $x - e^{z_i}$ is an optimal solution of the program

$$\max\{\sum_{j\in N_1\setminus Z} c_j w_j : w \in \{0,1\}^{N_1\setminus Z} : A_{1\cdot} w \le b_1 - a_{1z_i}\}.$$

This shows that in this case x satisfies inequality (8.6) as an equation, too. We obtain that $c^T x \le \gamma$ must be a facet of the knapsack polyhedron P^1. This completes the proof. □

The correctness of the theorem strongly relies on the fact that every pair of items from the intersection $N_1 \cap N_2$ is a cover. If this condition is not satisfied, then Example 8.4.3 demonstrates that the facet-defining inequalities of the two single knapsack polyhedra do not suffice in general to describe the polyhedron associated with the intersection of the two knapsack constraints. Geometrically, the fact that we intersect two knapsack constraints generates incomparabilities between the column vectors of the associated matrix. These incomparabilities give rise to cutting planes that do not define facets of one of the two single knapsack polyhedra that we intersect. In fact, incomparabilities between column vectors in a matrix make it possible to "melt" inequalities from different knapsack polyhedra. A basic situation to which the operation of melting applies is

Proposition 8.4.5 *Let $m = 2$, $A \in \mathbb{N}^{2\times n}$ and $Ax \leq b$ be a system of two consecutively intersecting knapsack constraints. Let $\sum_{i\in N_1\setminus N_2} c_i x_i + \sum_{i\in N_1\cap N_2} c_i x_i \leq \gamma$ be a valid inequality for P^1, and let $\sum_{i\in N_2\setminus N_1} c_i x_i + \sum_{i\in N_1\cap N_2} c_i x_i \leq \gamma$ be a valid inequality for P^2. Setting $\Theta := \sum_{i\in N_1\setminus N_2} c_i$, the* melted inequality

$$\sum_{i\in N_1\setminus N_2} c_i x_i + \sum_{i\in N_1\cap N_2} c_i x_i + \sum_{i\in N_2\setminus N_1} (c_i - \Theta)^+ x_i \leq \gamma$$

is valid for P_{IsK}.

Proof. Let $x \in P_{\mathrm{IsK}} \cap \mathbb{Z}^N$. If $x_i = 0$ for all $i \in N_2 \setminus N_1$ with $c_i - \Theta > 0$, the inequality is satisfied because $\sum_{i\in N_1\setminus N_2} c_i x_i + \sum_{i\in N_1\cap N_2} c_i x_i \leq \gamma$ is valid for P^1. Otherwise, $\sum_{i\in N_2\setminus N_1} (c_i - \Theta)^+ x_i \leq \sum_{i\in N_2\setminus N_1} c_i x_i - \Theta$, and we obtain

$$\begin{aligned}
\sum_{i\in N_1\setminus N_2} c_i x_i \quad &+ \sum_{i\in N_1\cap N_2} c_i x_i + \sum_{i\in N_2\setminus N_1} (c_i - \Theta)^+ x_i \\
&\leq \sum_{i\in N_1\setminus N_2} c_i x_i - \Theta + \sum_{i\in N_1\cap N_2} c_i x_i + \sum_{i\in N_2\setminus N_1} c_i x_i \\
&\leq \sum_{i\in N_1\cap N_2} c_i x_i + \sum_{i\in N_2\setminus N_1} c_i x_i \\
&\leq \gamma.
\end{aligned}$$

□

Proposition 8.4.5 can be extended to general upper bounds $u \in \mathbb{N}^n$. Often the inequality that results from melting valid inequalities from knapsack polyhedra as described in Proposition 8.4.5 does not define a facet of P_{IsK}. In such situations there is a good chance of strengthening the melted inequality by determining lifting coefficients for the items in $(N_1 \setminus N_2) \cup (N_2 \setminus N_1)$ with respect to a given order.

Example 8.4.6 *Let A be the matrix $\begin{bmatrix} 1 & 4 & 5 & 6 & 0 & 0 \\ 0 & 0 & 5 & 6 & 1 & 4 \end{bmatrix}$ and $b = \begin{bmatrix} 15 \\ 15 \end{bmatrix}$. The inequality $x_1 + 4x_2 + 5x_3 + 6x_4 \leq 15$ defines a facet of $P^1 := \mathrm{conv}\{x \in \mathbb{Z}^4 : x_1 + 4x_2 + 5x_3 + 6x_4 \leq 15,\ 0 \leq x_i \leq 3,\ i \in \{1,2,3,4\}\}$. For $\alpha \in \{0,1,2,3\}$ the inequality $5x_3 + 6x_4 + \alpha x_6 \leq 15$ is valid for $P^2 := \mathrm{conv}\{x \in \mathbb{Z}^{\{3,4,5,6\}} : 5x_3 + 6x_4 + x_5 + 4x_6 \leq 15,\ 0 \leq x_i \leq 3,\ i \in \{3,4,5\}, 0 \leq x_6 \leq 1\}$. Setting $\alpha = 1$ and applying Proposition 8.4.5 and the succeeding remark we obtain that $0x_1 + 3x_2 + 5x_3 + 6x_4 + x_6 \leq 15$ is valid for $P_{IsK} := \mathrm{conv}\{x \in \mathbb{Z}^6 : Ax \leq b, 0 \leq x_i \leq 3,\ i \in \{1,2,3,4,5\}, 0 \leq x_6 \leq 1\}$. This inequality can be strengthened by lifting to yield the facet-defining inequality $x_1 + 3x_2 + 5x_3 + 6x_4 + x_6 \leq 15$. For $\alpha = 2$, we end up with the melted inequality $0x_1 + 2x_2 + 5x_3 + 6x_4 + 2x_6 \leq 15$. For $\alpha = 3$ we obtain the inequality $0x_1 + x_2 + 5x_3 + 6x_4 + 3x_6 \leq 15$ that can again be strengthened to yield a facet-defining inequality of P_{IsK}, $x_2 + 5x_3 + 6x_4 + x_5 + 3x_6 \leq 15$.*

It turns out that sometimes the feasible set inequalities for two consecutively intersecting knapsacks can be interpreted in terms of melting feasible set inequalities for the associated single knapsack polytopes.

Example 8.4.7 *Consider $A = \begin{bmatrix} 15 & 3 & 5 & 13 & 0 & 0 \\ 0 & 17 & 18 & 17 & 19 & 20 \end{bmatrix}$ and $b = \begin{bmatrix} 20 \\ 35 \end{bmatrix}$. The set $T = \{2,3\}$ is feasible with respect to weights $w_2 = w_3 = 1$, and the inequality $x_2 + x_3 + x_4 + 2x_5 + 2x_6 \leq 2$ is a feasible set inequality for $P^2 := \mathrm{conv}\{x \in \{0,1\}^{\{2,3,4,5,6\}} : 17x_2 + 18x_3 + 17x_4 + 19x_5 + 20x_6 \leq 35\}$, where the variables not in T are lifted in the sequence 4, 5, 6. In addition, $x_1 + x_2 + x_3 + x_4 \leq 2$ is a feasible* *set inequality for* $P^1 := \text{conv}\{x \in \{0,1\}^{\{1,2,3,4\}} : 15x_1 + 3x_2 + 5x_3 + 13x_4 \leq 20\}$, *with respect to the same feasible set* T *and the lifting sequence* 4, 1. *Now* $\Theta = \sum_{i\in N_1\setminus N_2} c_i = 1$ *and the melted inequality reads* $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 2$, *which is facet-defining for* $P_{IsK} := \{x \in \{0,1\}^6 : Ax \leq b\}$. *Note that the melted inequality is also a feasible set inequality with respect to* T *and* w_T *and the lifting sequence* 4, 1, 5, 6.

Observe that Example 8.4.7 also shows that under certain conditions the operation of melting feasible set inequalities produces a facet-defining inequality for P_{IsK}.

8.5 Separating Feasible Set Inequalities

When it comes to incorporate feasible set inequalities into a general mixed integer programming solver like `SIP` many difficulties and questions arise. First, how to find a feasible set T? Consider some general mixed integer program

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \leq b \\ & l \leq x \leq u \\ & x \in \mathbb{Z}^N \times \mathbb{R}^C, \end{array} \tag{8.7}$$

where N, C and M are finite sets, N and C disjoint, $A \in \mathbb{R}^{M\times(N\cup C)}$, and l, u, c and b are vectors of appropriate dimension. In general, the given constraint matrix A contains positive and negative entries, and it is a-priori not clear, how to find some subset $T \subseteq N$ such that u_T is feasible for (8.7). Second, once one determined a set T, what is a suitable weighting w for T as proposed in Definition 8.1.1? In principle, every vector $w \in \mathbb{Z}^T$ is possible. Third, how to perform the lifting? The calculation of the exact lifting coefficient for some variable x_j requires the solution of u_j many integer programs, which might be very expensive to compute. Fourth, what are good substructures of A to start with and look for feasible set inequalities. It is pointless to begin with the whole constraint matrix A, since in the same time we find one cutting plane we probably would solve the entire problem. Desirable would be to have the matrix A in bordered block diagonal form as discussed in Chapter 6, since blocks in this form are independent of each other and, assuming they are small, we might even perform the lifting exactly. Questions over questions.

It goes without saying that most of these questions can only be answered by computational experiences. We discuss some of the options below. Some properties of feasible set inequalities, however, leave us no choice and restrict our search for good substructures of A. Let us call a substructure of A in analogy to Chapter 6 a *block*. One such property of a feasible set inequality is that it is defined for integer variables with finite lower and upper bounds only. Thus, we remove from our considerations all rows that contain continuous variables or variables with infinite lower or upper bound. Furthermore, we can restrict ourselves to fractional variables. If we are faced with some block containing only variables whose current LP values are integer, there cannot exist a violated feasible set inequality. Thus, in our choice of the block we only take fractional variables into account. The remaining, currently integer variables, will be lifted afterwards. This fact has an important advantage, since the number of fractional variables is usually only some fraction of the total number of variables (often below 10%), and restricts our search for possible blocks to a small part of A.

In detail, we proceed as follows in order to find a reasonable substructure $Bx \leq d$ of $Ax \leq b$, i.e., subsets $I \subseteq M$, $J \subseteq N$ with $B = A_{I,J}$ and $d = b_J$. As outlined possible candidates for I are only rows that solely contain integer variables with finite lower and upper bounds and that contain at least one variable whose current LP value is fractional. Second, we restrict ourselves to rows whose number of non-zeros lies between three and some upper bound σ. 'Three' is not really a restriction, since if some row has only two non-zero entries a complete description of the associated integer knapsack polytope is readily available, see Section 5.4. The number σ is a parameter, which basically is used to control the size of the final block. We have experimented with this parameter a lot, and a reasonable setting on our test set is $\sigma = 20$. We also exclude rows from further consideration that are not tight for the current LP solution. Let us denote by $I_c \subseteq M$ the set of candidate rows. We now construct the row intersection graph, see also page 109, i. e., the graph that is obtained by introducing a node for each row in I_c and an edge whenever two rows intersect in some fractional column. It turns out that due to the fact that usually the number of fractionals is small the row intersection graph for most problems decomposes into components of small size. 'Small' here means no more than three/four rows and up to thirty/forty variables per component. For only very few instances, in particular the "stein"-problems and problem seymour, the row intersection graph contains big components. In this case, we decompose the big components heuristically. Motivated by the success of the greedy-type algorithms in Chapter 6 we start with a row of a component that has largest degree and iteratively add further rows in order of decreasing intersection with the already selected rows. Our current setting allows at most ten rows or around fifty variables in a block. If one of the limits is reached we stop.

After applying this procedure we obtain index sets I_1 and J_1 that define a substructure $Bx \le d$ of $Ax \le b$ that contains at most ten rows and up to fifty variables. We delete the rows from I_c whose support intersects with J_1. With the remaining rows we apply the same procedure as described, and we end with mutually disjoint sets $I_1, \dots, I_\beta \subseteq M$ and mutually disjoint sets $J_1, \dots, J_\beta \subseteq N$. For each block $A_{I_b,J_b} x_{J_b} \le b_{I_b}$ we call our separation algorithm to derive violated feasible set inequalities. The following algorithm outlines our procedure and provides our answers to the questions raised at the beginning of the section. An explanation of the steps will be given afterwards.

Algorithm 8.5.1 *Separation algorithm for feasible set inequalities.*

Input: *A block $A_{I,J} x_J \le b_I$, $l_J \le x_J \le u_J$ of $Ax \le b$, $l \le x \le u$ with $I \subseteq M$ and $J \subseteq N$; an optimal solution $\bar{x}$ of the current LP.*

(1) Fix all variables that are integer.

$$f_i = \begin{cases} \bar{x}_i, & \text{if } \bar{x}_i \in \mathbb{Z}; \\ FREE, & \text{if } i \in \mathrm{Fr}(\bar{x}) := \{j \in J : \bar{x}_j \notin \mathbb{Z}\}. \end{cases}$$

(2) Solve

$$(8.8) \qquad \begin{array}{ll} \max & \sum_{i \in \mathrm{Fr}(\bar{x})} \frac{\bar{x}_i}{u_i} z_i \\ & \sum_{i \in \mathrm{Fr}(\bar{x})} A_{\cdot i} z_i \le b - \sum_{i \notin \mathrm{Fr}(\bar{x})} A_{\cdot i} f_i \\ & l_{\mathrm{Fr}(\bar{x})} \le z \le u_{\mathrm{Fr}(\bar{x})},\ z \in \mathbb{Z}^{\mathrm{Fr}(\bar{x})} \end{array}$$

(3) **If** *(8.8) is infeasible, try to derive a violated* bound inequality.
Else *let y be an optimal solution of (8.8).*
Set $T := \mathrm{supp}(y)$.

(4) Assign weights w_j for $j \in T$ according to

$$w_j = \begin{cases} +1, & \text{if } |\{i \in I : a_{ij} > 0\}| \ge |\{i \in I : a_{ij} < 0\}|; \\ -1, & \text{else,} \end{cases}$$

and set $f_i := l_i$ for $i \in \mathrm{Fr}(\bar{x}) \setminus T$.

(5) Solve

$$\text{(8.9)} \qquad \begin{array}{rl} \omega = \max & \sum_{i \in T} w_i z_i \\ & \sum_{i \in T} A_{\cdot i} z_i \leq b - \sum_{i \in J \setminus T} A_{\cdot i} f_i \\ & l_T \leq z \leq u_T,\ z \in \mathbb{Z}^T. \end{array}$$

(6) $\sum_{i \in T} w_i x_i \leq \omega$ *is the basic feasible set inequality.*

(7) Determine a lifting sequence $\pi_1, \dots, \pi_{|J|-|T|}$ *of* $J \setminus T$ *such that all fractional variables come first, i. e.,* $\{\pi_1, \dots, \pi_{|\mathrm{Fr}(\bar{x}) \setminus T|}\} = \mathrm{Fr}(\bar{x}) \setminus T$.

(8) **For** $k = 1, \dots, |J| - |T|$ *perform the following steps.*
Determine lifting coefficient w_{π_k} *for variable* x_{π_k}.
If $k > |\mathrm{Fr}(\bar{x}) \setminus T|$ *and the current feasible set inequality is not violated –* **Stop** *(no violated inequality found).*

(9) Check whether the lifted feasible set inequality $w^T x \leq \omega$ *is violated.*

(10) **Stop.**

Steps (2) through (5) of Algorithm 8.5.1 show our choice of determining a "feasible" set T. The integer program (8.8) is solved to optimality using `SIP`. Note that as outlined above the number of fractional variables is usually very small resulting in integer programs of small size. Thus, the expected time to solve this integer program is small, and our computational results in the next section confirm this. For the integer program (8.9) we supply a limit on the number of branch-and-bound nodes of 1000. If we are not able to solve the problem within this limit we use $\omega = \lfloor \hat{\omega} \rfloor$, where $\hat{\omega}$ is the best upper bound after 1000 branch-and-bound nodes. We played with several different objective functions in (8.8), the one here performed best on average. It also has the advantage that it coincides with the starting set chosen in Algorithm 5.3.1, where the constraint system defines a 0/1 knapsack problem. Note that the set T as defined in Step (3) need not be "feasible" with respect to the weighting in Step (4) as required in Definition 8.1.1, since it is not guaranteed that u_T is a feasible solution and that $\omega \leq \sum_{i \in T} w_i u_i$. However, given the set T and weights $w_i, i \in T$, (8.9) determines the best possible right-hand side such that $\sum_{i \in T} w_i x_i \leq \omega$ is valid for $P_{\mathrm{IsK}}(T)$. Although the set T and w_T might not satisfy the requirements of Definition 8.1.1, we call it still *feasible* in the following. Our weighting w is restricted to take ± 1 values only. We also tried different weightings with limited success. The setting given in Step (4) is again motivated by the fact that it coincides with the 0/1 knapsack case. Note also that if (8.8) does not have a feasible solution, see Step (3), and if all currently integer variables are fixed at their lower or upper bounds, i. e., $f_i \in \{l_i, u_i\}$ for all $i \notin \mathrm{Fr}(\bar{x})$, the inequality

$$\text{(8.10)} \qquad \sum_{\{i : f_i = l_i\}} (x_i - l_i) + \sum_{\{i : f_i = u_i\}} (u_i - x_i) \geq 1,$$

called *bound inequality*, is valid for $P = \mathrm{conv}\{x \in \mathbb{R}^{N \cup C} : x \text{ feasible for (8.7)}\}$. This inequality is also violated by the current LP solution $\bar{x}$ and we return with this violated inequality from Algorithm 8.5.1.

It remains to discuss the lifting, Steps (7) and (8). The lifting sequence starts with all remaining fractional variables, sorted in non-increasing order of their LP value, followed by all integer variables that are not at one of their bounds, i. e., $l_i < f_i < u_i$, and last, we lift all integer variables that are currently at one of their bounds. Within these two latter sets we specify no particular order. The lifting of one variable basically follows the scheme described in Algorithm C.2 with some adjustments. First, we have to take into account in (C.2) that certain variables are fixed to some non-zero value. This can easily be achieved as outlined below. A more serious problem is the lifting of variables that are currently neither fixed on their lower nor on their upper bound. In this case, it might be possible that the lifted inequality is no longer valid. Before discussing this difficulty in detail, let us describe the overall algorithm.

Algorithm 8.5.2 *Lifting variable* x_{π_k}.

Input: *An inequality* $\sum_{i\in S} w_i x_i \le \omega$ *valid for* $P_{IsK}(I, S, A_{I,S}, b - \sum_{i\in N\setminus S} A_{\cdot i} f_i, u_S)$, *where* $S := T \cup \{\pi_1, \dots, \pi_{k-1}\}$.

Output: *Lifting coefficient* w_{π_k} *for* x_{π_k} *and (new) right-hand side* ω *such that*

$$\sum_{i\in S\cup\{\pi_k\}} w_i x_i \le \omega \tag{8.11}$$

is valid for $P_{IsK}(I, S\cup\{\pi_k\}, A_{I,S\cup\{\pi_k\}}, b - \sum_{i\in N\setminus(S\cup\{\pi_k\})} A_{\cdot i} f_i, u_{S\cup\{\pi_k\}})$.

(1) **For** $l = l_i, \dots, u_i$ *compute*

$$\begin{array}{rl} \omega_l = \max & \sum_{i\in S} w_i z_i \\ & \sum_{i\in S} A_{\cdot i} z_i + A_{\cdot \pi_k}\cdot l \le b - \sum_{i\in N\setminus S} A_{\cdot i} f_i \\ & 0 \le z \le u_S,\ z \in \mathbb{Z}^S. \end{array} \tag{8.12}$$

(2) *Let*

$$\begin{array}{rcl} w^- &=& \displaystyle\max_{l=l_{\pi_k},\dots,f_{\pi_k}-1} \frac{\omega-\omega_l}{l-f_{\pi_k}} \\ w^+ &=& \displaystyle\min_{l=f_{\pi_k}+1,\dots,u_{\pi_k}} \frac{\omega-\omega_l}{l-f_{\pi_k}} \end{array} \tag{8.13}$$

(3) **If** $f_{\pi_k} = l_{\pi_k}$ *set*

$$\begin{array}{ll} w_{\pi_k} & = w^+ \\ \omega & = \omega + w^+ f_{\pi_k} \end{array} \tag{8.14}$$

(4) **If** $f_{\pi_k} = u_{\pi_k}$ *set*

$$\begin{array}{ll} w_{\pi_k} & = w^- \\ \omega & = \omega + w^- f_{\pi_k} \end{array} \tag{8.15}$$

(5) **If** $l_{\pi_k} < f_{\pi_k} < u_{\pi_k}$ *perform the following steps:*
If $w^- \le w^+$ *return with (8.14) or (8.15).*
If $w^- > w^+$ *return with*

$$\begin{array}{ll} w_{\pi_k} & = w^- \\ \omega & = \omega + w^- f_{\pi_k} + (u_{\pi_k} - f_{\pi_k})(w^- - w^+) \end{array} \tag{8.16}$$

or with

$$\begin{array}{ll} w_{\pi_k} & = w^+ \\ \omega & = \omega + w^- f_{\pi_k} - l_{\pi_k}(w^- - w^+) \end{array} \tag{8.17}$$

or with

$$\begin{array}{ll} w_{\pi_k} & = \dfrac{u_{\pi_k} - l_{\pi_k} - f_{\pi_k}}{u_{\pi_k} - l_{\pi_k}}(w^+ - w^-) + w^- \\ \omega & = \omega + w^- f_{\pi_k} - \dfrac{u_{\pi_k} - l_{\pi_k} - f_{\pi_k}}{u_{\pi_k} - l_{\pi_k}}(w^+ - w^-) l_{\pi_k}, \end{array} \tag{8.18}$$

the latter is only valid if $l_{\pi_k} + f_{\pi_k} \le u_{\pi_k}$.

(6) **Stop.**

The integer program (8.7) coincides with the integer program in (C.2) with the only difference that the right-hand side in (8.12) is adjusted by the fixings of the variables coming later in the sequence. Accordingly, the computation of w^+ in (8.13) coincides with (C.3). The following proposition shows that the settings in Steps (3) and (4) of Algorithm 8.5.2 are correct.

Proposition 8.5.3

(a) If $f_{\pi_k} \in \{l_{\pi_k}, u_{\pi_k}\}$, the settings in (8.14) and (8.15) yield a valid inequality.

(b) If $l_{\pi_k} < f_{\pi_k} < u_{\pi_k}$ and $w^- \leq w^+$, both (8.14) and (8.15) yield a valid inequality.

Proof. Let $\bar{x} \in P_{\text{IsK}}(I, S \cup \{\pi_k\}, A_{I,S\cup\{\pi_k\}}, b - \sum_{i \in N\setminus(S\cup\{\pi_k\})} A_{\cdot i} f_i, u_{S\cup\{\pi_k\}})$ be integer.

Consider part (a) and the case $f_{\pi_k} = l_{\pi_k}$ and denote by $\bar{\omega} = \omega + w^+ f_{\pi_k}$ the new right-hand side. Note that the lifted inequality (8.11) can be rewritten as

$$\sum_{i\in S} w_i x_i + w_{\pi_k}(x_{\pi_k} - l_{\pi_k}) \leq \omega$$

Thus, if $\bar{x}_{\pi_k} = l_{\pi_k}$, the inequality is trivially valid. If $\bar{x}_{\pi_k} = l$ for some $l_{\pi_k} < l < u_{\pi_k}$, we obtain $\sum_{i\in S} w_i \bar{x}_i + w_{\pi_k}(\bar{x}_{\pi_k} - l_{\pi_k}) \leq \sum_{i\in S} w_i \bar{x}_i + \frac{\omega - \omega_l}{l - l_{\pi_k}}(l - l_{\pi_k}) \leq \omega_l + (\omega - \omega_l) = \omega$. The same proof applies if $f_{\pi_k} = u_{\pi_k}$.

We show (b) for the settings in (8.14), i.e., we prove validity of $\sum_{i\in S} w_i x_i + w^+(x_{\pi_k} - f_{\pi_k}) \leq \omega$. (8.15) can be shown analogously. If $\bar{x}_{\pi_k} \in \{f_{\pi_k}, \ldots, u_{\pi_k}\}$ the validity follows by part (a). If $\bar{x}_{\pi_k} \in \{l_{\pi_k}, \ldots, f_{\pi_k} - 1\}$, we get $\sum_{i\in S} w_i \bar{x}_i + w^+(\bar{x}_{\pi_k} - f_{\pi_k}) \leq \sum_{i\in S} w_i x_i + w^-(x_{\pi_k} - f_{\pi_k}) \leq \omega$. The first inequality follows from $\bar{x}_{\pi_k} - f_{\pi_k} < 0$ and $w^- > w^+$, the second from part (a). □

It remains to discuss the case $l_{\pi_k} < f_{\pi_k} < u_{\pi_k}$ and $w^- > w^+$. In this case none of the two settings (8.14) and (8.15) needs to be valid as the following example shows.

Example 8.5.4 *Consider the integer program*

$$\begin{array}{rl} \min & 4200\,x_1 + 1500\,x_2 + 2700\,x_3 \\ & 0.9\,x_1 + \quad x_2 - \quad x_3 \quad = 5.7 \\ & 0 \leq x_1 \leq 15 \\ & 0 \leq x_2 \leq 18 \\ & 0 \leq x_3 \leq 18,\ x \in \mathbb{Z}^3. \end{array}$$

This is a substructure of example flugpl *from the* `Miplib`. *Running* `SIP` *for this problem yields an optimal solution for an LP encountered during the solution process, where the three variables have values $x_1 = 5.5, x_2 = 3.75$, and $x_3 = 3$. Thus we fix variable x_3 to three, i.e., $f_3 = 3$. The inequality $x_1 + x_2 \leq 9$ is valid for $\text{conv}\{x \in \mathbb{Z}^2 : 0.9x_1 + x_2 = 8.7,\ 0 \leq x_1 \leq 15,\ 0 \leq x_2 \leq 18\}$. Now in order to lift x_3 we compute w^+ and w^- in (8.13). We get $w^+ = -\frac{4}{3}$ and the associated inequality (8.14) reads $x_1 + x_2 - \frac{4}{3}x_3 \leq 5$. It is violated by the feasible solution $x = (3, 3, 0)^T$. Similarly, $w^- = -1$ and the inequality $x_1 + x_2 - x_3 \leq 6$ from (8.15) is violated by the feasible solution $x = (13, 0, 6)^T$.*

The problem discussed in Example 8.5.4 is not new. In fact, Balas, Ceria, Cornuéjols, and Natraj [1996] give another example in conjunction with Chvátal-Gomory cuts. Here the same difficulty arises, when Chvátal-Gomory cuts are generated at some node, which is not the root note. When integer variables are involved these inequalities are usually only locally valid for this node and its descendants, but not for the entire branch-and-bound tree. Balas, Ceria, Cornuéjols, and Natraj [1996] show that cuts are indeed globally valid if all integer variables are binary.

Algorithm 8.5.2 gives one possible answer to the general integer case. The idea is to consider two polytopes P^+ and P^-. P^+ is the original polytope intersected with the inequality using formula (8.14) and the bound constraints $f_{\pi_k} \leq x_{\pi_k} \leq u_{\pi_k}$. P^- is defined accordingly, i. e.,

$$\begin{aligned} P^- &= \{x \in P_{\text{IsK}} : \textstyle\sum_{i\in S} w_i x_i + w^- x_{\pi_k} \leq \omega + w^- f_{\pi_k}, l_{\pi_k} \leq x_{\pi_k} \leq f_{\pi_k}\}. \\ P^+ &= \{x \in P_{\text{IsK}} : \textstyle\sum_{i\in S} w_i x_i + w^+ x_{\pi_k} \leq \omega + w^+ f_{\pi_k}, f_{\pi_k} \leq x_{\pi_k} \leq u_{\pi_k}\}. \end{aligned} \tag{8.19}$$

Now we can apply the idea of disjunctive programming and determine $\text{conv}(P^- \cup P^+)$ to derive a valid inequality for $P_{\text{IsK}}(S \cup \{\pi_k\})$.

Let us briefly summarize the concept of disjunctive programming applied to our case, for more details see Balas, Ceria, and Cornuéjols [1993] and the references therein. Suppose for the ease of exposition that $P^+ = \{x \in \mathbb{R}^N : A^1 x \leq b^1\}$ and $P^- = \{x \in \mathbb{R}^N : A^2 x \leq b^2\}$. Now

$$\begin{aligned} \text{conv}(P^- \cup P^+) = \{x \in \mathbb{R}^N : \; & \text{there exist } \bar{y}^1, \bar{y}^2, \lambda^1, \lambda^2 \text{ such that:} \\ & (a)\;\; x = \lambda^1 \bar{y}^1 + \lambda^2 \bar{y}^2 \\ & (b)\;\; A^1 \bar{y}^1 \leq b^1 \\ & (c)\;\; A^2 \bar{y}^2 \leq b^2 \\ & (d)\;\; \lambda^1 + \lambda^2 = 1, \lambda^1 \geq 0, \lambda^2 \geq 0\}. \end{aligned} \tag{8.20}$$

(8.20) (a) contains quadratic terms. Multiplying (8.20) with λ_i and setting $y^i := \lambda_i \bar{y}^i$ for $i = 1, 2$ yields the polyhedron

$$\begin{aligned} Q = \{x \in \mathbb{R}^N : \; & \text{there exist } y^1, y^2, \lambda^1, \lambda^2 \text{ such that:} \\ & (a)\;\; x = y^1 + y^2 \\ & (b)\;\; A^1 y^1 \leq \lambda^1 b^1 \\ & (c)\;\; A^2 y^2 \leq \lambda^2 b^2 \\ & (d)\;\; \lambda^1 + \lambda^2 = 1, \lambda^1 \geq 0, \lambda^2 \geq 0\}. \end{aligned} \tag{8.21}$$

Q still coincides with $\text{conv}(P^- \cup P^+)$ if we guarantee that $y^i = 0$ whenever $\lambda_i = 0$. This is true, since P^+ and P^- are bounded implying that $\{y^i \in \mathbb{R}^n : Ay^i \leq 0\} = \{0\}$ for $i = 1, 2$.

Let us abbreviate the set of vectors $x, y^1, y^2, \lambda^1, \lambda^2$ satisfying (8.21) (a) – (d) by $L = \{(x,z)^T : Dx + Bz \leq d\}$, where $z = (y^1, y^2, \lambda^1, \lambda^2)^T$ and B, D, d are appropriate matrices and vectors. Q, and $\text{conv}(P^- \cup P^+)$, is the projection of L onto the x-space, i. e.,

$$\text{conv}(P^- \cup P^+) = \{x \in \mathbb{R}^N : \text{there exists } z \text{ such that } (x,z)^T \in L\}.$$

In order to obtain from L a description of $\text{conv}(P^- \cup P^+)$ by means of linear inequalities, we need to eliminate the z-variables. With

$$C = \{v : \; v^T B = 0, \; v \geq 0\} \tag{8.22}$$

we get

$$\text{conv}(P^- \cup P^+) = \{x \in \mathbb{R}^N : (v^T D)x \leq v^T d \text{ for all } v \in C\}.$$

In order to find a valid inequality that cuts off the LP solution $\bar{x}$ we solve the linear program

$$\begin{aligned} \max \quad & (D\bar{x} - d)^T v \\ & v \in C. \end{aligned} \tag{8.23}$$

Note that if there is a violated inequality in C, then (8.23) is unbounded, since C is a polyhedral cone. For algorithmic conveniences C is often truncated by bounding the vector v with respect to some norm. Solving the linear program in (8.23) yields an inequality that is valid for P_{IsK}. If the optimum in (8.23) is zero, there exists no violated inequality, otherwise $(\bar{v}^T D)x \leq \bar{v}^T d$ yields the desired inequality, where $\bar{v}$ is an optimal solution (extreme ray) of (8.23).

We apply this scheme in our implementation for the following two polytopes

$$\begin{array}{ll}(8.24) & \begin{array}{l}\tilde{P}^- = \{x \in \mathbb{R}^N : \sum_{i \in S} w_i x_i + w^- x_{\pi_k} \leq \omega + w^- f_{\pi_k}, l_{\pi_k} \leq x_{\pi_k} \leq f_{\pi_k}\}, \\ \tilde{P}^+ = \{x \in \mathbb{R}^N : \sum_{i \in S} w_i x_i + w^+ x_{\pi_k} \leq \omega + w^+ f_{\pi_k}, f_{\pi_k} \leq x_{\pi_k} \leq u_{\pi_k}\}.\end{array}\end{array}$$

That is, we consider a relaxation of the polytopes defined in (8.19) by neglecting the constraints defining P_{IsK}. The advantage of using this relaxation is that – as we will see in a moment – we can solve the resulting linear program (8.23) explicitly. The drawback, of course, is that we might get a weaker inequality. The linear program (8.23) that results when using $\tilde{P}^-$ and $\tilde{P}^+$ in (8.24) has (after some obvious variable substitutions) seven variables and two constraints. We fix the variable v_i to one that corresponds to the constraint $\sum_{i \in S} w_i x_i + w^- x_{\pi_k} \leq \omega + w^- f_{\pi_k}$. This way we guarantee that the coefficients w_i for $i \in S$ stay unchanged in the resulting lifted inequality. After doing some calculations it turns out that this linear program, with now six variables and two constraints, has only three possible optimal solutions. The three LP solutions yield three different inequalities as stated in (8.16), (8.17), and (8.18). We use the one that maximizes the slack.

8.6 Computational Results

In this section we report on our computational experience with Algorithm 8.5.2 for the feasible set inequalities. We have incorporated this algorithm in `SIP`, and tested it on the instances from the `Miplib`. We compared `SIP` with the default parameter setting and `SIP` where in addition the separation algorithm for feasible set inequalities has been turned on. As in all other test runs, we use a time limit of 3600 CPU seconds and limit the number of branch-and-bound nodes to one million.

It turns out that for 48 of the problems we do not find feasible set inequalities and the overhead for applying our separation algorithm is below 1% of the total running time. For the remaining 11 problems we find feasible set inequalities or our separation routine uses more than 1% of the computation time. Tables 8.1 and 8.2 show the results for these examples. Column 1 gives the problem name followed by the number of branch-and-bound nodes in Column 2. The next two columns give the number of cuts found, *Others* include the 0/1 knapsack inequalities with or without generalized upper bounds (see Section 5.3), and the mixed integer weight inequalities (see Section 5.4). Column *FS* give the number of feasible set inequalities found. Columns 5 and 6 show timings, the time used in Algorithm 8.5.2 and the total running time. If we cannot solve the problem within the time limit of 3600 seconds, the last Column *Gap %* shows a non-zero gap ($= 100.0 \frac{|\text{upper bound - lower bound}|}{|\text{lower bound}|}$).

From Table 8.2 we see that we can significantly reduce the gap for test problem seymour and slightly for gesa2 and p2756, resulting in total reduction of about 25%. We also recognize that basically no time is spent for separating feasible set inequalities. This is very astonishing, since almost all integer programs that come up in Algorithm 8.5.2 (and these are thousands) are solved to optimality, where we use an upper bound on the branch-and-bound nodes of 1000. The success of the feasible set inequalities relies on the fact that we *can* solve small integer programs to optimality in general very fast. There are other interesting facts that can be read from Table 8.2. The "stein"-problems, where we find a considerable amount of cuts, and

Example	B & B	Cuts		Time		Gap %
		Others	FS	FS	Total	
fiber	783	372	0	0.0	16.9	0.000
gesa2	209525	33	0	0.0	3600.0	0.048
gesa3_o	74472	0	0	0.0	1144.7	0.000
misc03	699	14	0	0.0	4.1	0.000
misc07	35585	0	0	0.0	378.8	0.000
p0033	57	32	0	0.0	0.1	0.000
p0201	507	136	0	0.0	5.0	0.000
p2756	23151	6923	0	0.0	3600.2	0.891
seymour	1947	0	0	0.0	3601.8	7.770
stein27	4666	0	0	0.0	8.0	0.000
stein45	54077	0	0	0.0	277.7	0.000
Total (11)	405469	7510	0	0.0	12637.3	8.709

Table 8.1: SIP without feasible set inequalities

Example	B & B	Cuts		Time		Gap %
		Others	FS	FS	Total	
fiber	771	358	1	3.3	19.4	0.000
gesa2	211168	33	4	9.0	3600.0	0.047
gesa3_o	74472	0	0	70.7	1218.4	0.000
misc03	703	14	11	1.1	5.4	0.000
misc07	20962	0	365	54.1	333.0	0.000
p0033	75	40	5	0.6	0.7	0.000
p0201	507	136	1	2.0	7.1	0.000
p2756	23041	6842	6	4.9	3600.9	0.803
seymour	2146	0	10	5.4	3600.0	5.726
stein27	3660	0	142	1.6	10.0	0.000
stein45	47612	0	783	23.6	308.5	0.000
Total (11)	385117	7423	1328	176.4	12703.4	6.576

Table 8.2: SIP with feasible set inequalities

example seymour, where we improve the quality substantially, are set covering problems. There seem to be virtually no efficient separation algorithms for set covering problems. To the best of our knowledge the only exceptions are the cutting planes from conditional bounds by Balas and Ho [1980], a class of k-projection inequalities by Nobili and Sassano [1992], and aggregated cycle inequalities by Borndörfer [1998]. It is therefore even more astonishing that our general separation algorithm for feasible set inequalities finds violated inequalities for this type of problems.

The only example where we spent a significant amount of time in Algorithm 8.5.2 and fail to find any violated cut is example gesa3_o. We looked at this example in detail and it turned out that the inequalities of the identified blocks already give a complete description of the polytope induced by this block. These means, that the fractional solution must be a convex combination of feasible integer solutions and we fail to find violated cuts. The question is, of course, how to avoid this case. One way to overcome this difficulty is to first check, whether the current fractional solution is a convex combination of feasible integer solutions, and then to start the search for violated cuts. This approach has been successfully used by Applegate, Bixby, Chvátal, and Cook [1998] for the solution of traveling salesman problems and there is good hope that their ideas might be carried over to our case, i. e., to general integer programs.

Chapter 9

Conclusions

In this thesis we studied integer programs with block structure. In Part I we looked at three different types of problems, the multiple knapsack problem, the Steiner tree packing problem and a special multicommodity flow problem, which led to integer programs with block structure. We focused our polyhedral studies on the impact that the linking constraints have on the associated polytopes. We observed that in general all individual inequalities are inherited by the packing polytope. In addition, we discovered many new classes of joint inequalities. The description of these inequalities and the conditions under which they are facet-defining are sometimes very complicated and technical, see, for instance, Theorem 2.4.2, Theorem 3.3.2, or Theorem 4.3.6. Also, from an algorithmic point of view, these inequalities are not tractable in general. For all discussed joint inequalities we only have separation heuristics at hand or our separation algorithms are restricted to special cases. Nevertheless, we have seen that these individual and especially the joint inequalities are indispensable for solving problems of realistic size.

Part II of this thesis dealt with general mixed integer programs. We discussed the main ingredients of a solver for such problems. As the application that led to the particular integer program under consideration is unknown to a general mixed integer programming algorithm, it is mandatory to analyze the underlying constraint matrix, extract known structures (like, for instance, the knapsack problem), and exploit knowledge about the detected structures in the solution process. We applied this methodology to block structures. In particular, we developed an algorithm that allows to identify block structure in a general mixed integer program. And indeed we saw that many real-world mixed integer programming problems do decompose into block structure. We then suggested two ways of exploiting block structure. One by parallelizing all column-based operations in the dual simplex algorithm. We saw that it is possible to get remarkable speed-ups for linear programs if the ratio of columns to rows is no more than ten (a ratio that is often satisfied by integer programs with block structure). In this context, it remains to parallelize the LU-factorization, a task whose solution integer programs with block structure would definitely benefit. The second, suggested way of exploiting block structure was a polyhedral one. We described a new, very general class of inequalities that is associated with the intersection of several inequalities. Such a set of inequalities may be viewed as one block and in the sense of Part I these inequalities correspond to the individual inequalities. As in Part I we saw that these individual inequalities do considerably improve the solution quality of general mixed integer programs. It remains the problem to identify joint inequalities in a general mixed integer program with block structure. The results of Part I show that this is a very difficult, but also a very worthwhile and highly promising challenge.

Appendix A

Notation

In this chapter we briefly summarize the notation we use throughout this thesis. In terms of polyhedral combinatorics we basically follow the book of Schrijver [1986] and Grötschel, Lovász, and Schrijver [1988], introductions to graph theory can be found, for instance, in Bondy and Murty [1976] and Chartrand and Lesniak [1986].

Graphs and Digraphs

We denote graphs by $G = (V, E)$, where V is the node set and E the edge set. All graphs we consider are finite. We denote an edge $e \in E$ with endnodes u and v by $[u, v]$ or uv for short. For a given edge set $F \subseteq E$, we denote by $V(F)$ all nodes that are incident to an edge in F. Given two node sets $U, W \subseteq V$, we denote by $[U : W]$ the set of edges in G with one endnode in U and the other in W. For a node set W, we also use $E(W)$ instead of $[W : W]$. By $G[W] = (W, E(W))$ we denote the graph induced by node set W. A set of node sets $V_1, \dots, V_p \subset V$, $p \geq 2$, is called a *partition of* V if all sets V_i are non-empty, the node sets are mutually disjoint and the union of these sets is V. (Note that we use "$\subset$" to denote strict set theoretic containment.) If $V_1, \dots, V_p$ is a partition of V then $\delta_G(V_1, \dots, V_p)$ denotes the set of edges in G whose endnodes are in different sets. For $W \subset V$, $W \neq \emptyset$, we write $\delta_G(W)$ instead of $\delta_G(W, V \setminus W)$ and call this set the *cut induced by* W. For two nodes $s, t \in V, s \neq t$, a cut $\delta_G(W)$, $W \subset V$, with $s \in W$ and $t \notin W$ is called an $[s, t]$-cut. If it is clear from the context we write $\delta(\cdot)$ instead of $\delta_G(\cdot)$. If $W = \{v\}$, we abbreviate $\delta(\{v\})$ by $\delta(v)$. For an edge set F, we define $d_F(v) = |\delta(v) \cap F|$ to be the degree of v in the subgraph (V, F) of G. For $v \in V$, $\eta(v)$ denotes all neighbours of v, i. e., $\eta(v) := V(\delta(v)) \setminus \{v\}$.

We call a sequence of nodes and edges $K = (v_0, e_1, v_1, e_2, \dots, v_{l-1}, e_l, v_l)$, where each edge e_i is incident with the nodes v_{i-1} and v_i for $i = 1, \dots, l$, and where the edges and nodes are pairwise disjoint (except possibly v_0 and v_l), a *path* (or a $[v_0, v_l]-$*path*), if $v_0 \neq v_l$, and a *cycle*, if $v_0 = v_l$ and $l \geq 2$. Each edge that connects two nodes of a cycle (path) K and that is not in K is called a *diagonal of* K. We say that two edges uv and $u'v'$ *cross with respect to* K if they appear in the sequence u, u', v, v' or u, v', v, u' by walking along the cycle (path). Similarly, we call two sets of diagonals F_1 and F_2 *cross free* if, for all $e_1 \in F_1$ and $e_2 \in F_2$, e_1 and e_2 do not cross. Otherwise, F_1 and F_2 are *crossing*. If not specified otherwise we consider a path P or a cycle C, respectively, as a subset of the edge set. We call an edge set B a *tree* if $(V(B), B)$ is connected and contains no cycle. The *leaves* of B are the nodes that are incident to exactly one edge of B. A graph $G = (V, E)$ is said to be k*-edge* (or k*-node*) *connected*, $k \geq 1$, if between any two nodes $u, v \in V$ there exist at least k edge- (or node-) disjoint $[u, v]$-paths.

A *flow* in an undirected graph $G = (V, E)$ with respect to some demand (s, t), $s, t \in V$ of size $d \in \mathbb{R}_+$ is a function $f : E \mapsto \mathbb{R}_+$ such that there exist $[s, t]$-paths $P_1, \ldots, P_p$, cycles $C_1, \ldots, C_q$, and positive numbers $\lambda_1, \ldots, \lambda_p, \mu_1, \ldots, \mu_q > 0$ such that

$$\text{(A.1)} \qquad \begin{aligned} f_{uv} &= \sum_{\{i:\, uv \in P_i\}} \lambda_i P_i + \sum_{\{i:\, uv \in C_i\}} \mu_i C_i \text{ for all } uv \in E, \\ d &= \sum_{i=1}^{p} \lambda_i. \end{aligned}$$

A *matching* M in a graph $G = (V, E)$ is a set of edges such that no two edges have a common endnode. The matching is called *perfect* if every node is adjacent to an edge in M.

We call a graph G a *complete rectangular* $h \times b$ *grid graph*, if it can be embedded in the plane by h horizontal lines and b vertical lines such that the nodes of V are represented by the intersections of the lines and the edges are represented by the connections of the intersections. A *column* J (*row* J) of a complete rectangular $h \times b$ grid graph is a subset of the edge set that has cardinality $h - 1$ $(b - 1)$ and whose edges correspond to the same vertical (horizontal) line.

A *digraph* $D = (V, A)$ consists of a finite set V of nodes and a finite set A of *arcs*. With each arc a an ordered tuple $(u, v) \in V \times V$ is associated, u is called the *tail* of a and v the head of a. All terminology of digraphs we need in this thesis is restricted to the following: For $v \in V$, $\delta^+(v) := \{(u, w) \in A : u = v\}$ denotes the set of all *outgoing* arcs of v and $\delta^-(v) := \{(u, w) \in A : w = v\}$ the set of all *ingoing arcs*; $|\delta^+(v)|$ is the *outdegree*, and $|\delta^-(v)|$ the *indegree of* v. Analogously to the undirected case, we define a *path (cycle)*, where in the definition of K edges are replaced by arcs. If in addition each arc a_i $(i = 1, \ldots, l)$ is of the form $a_i = (v_{i-1}, v_i)$, the path (cycle) is called *directed*. If D does not contain a directed cycle, it is called *acyclic*.

Vectors

Vectors are considered as column vectors unless otherwise specified. The superscript "T" denotes transposition. Consider some finite set N. We denote by $\mathbb{R}^N$ the $|N|$-dimensional vector space where the components of each vector are indexed by the elements of N, i. e., $x = (x_i)_{i \in N}$ for $x \in \mathbb{R}^N$. For $x \in \mathbb{R}^N$ and $I \subseteq N$, we will often abbreviate $\sum_{i \in I} x_i$ by $x(I)$. For $I \subseteq N$, we denote by χ^I the *incidence vector* of I in $\mathbb{R}^N$, i. e., $\chi_i^I = 1$, if $i \in I$, and $\chi_i^I = 0$, otherwise.

For $v \in \mathbb{R}^N$, we denote by v^+ the vector with $v_i^+ = v_i$ if $v_i \geq 0$ and $v_i^+ = 0$, otherwise. Accordingly, v^- is the vector with $v_i^- = -v_i$ if $v_i \leq 0$ and $v_i^- = 0$, otherwise. Clearly $v = v^+ - v^-$. If $v \in \mathbb{R}^N$, then $\operatorname{supp}(v)$ denotes the support of v; in formulas, $\operatorname{supp}(v) := \{i \in N : v_i \neq 0\}$.

The vector $e_i \in \mathbb{R}^N, i \in N$ denotes the i-th unit vector, i. e., the vector with a value of one in the i-th component and zero otherwise. The vector of all ones is denoted by $\mathbb{1}$.

Consider some matrix $A \in \mathbb{R}^{M \times N}$, where N, M are finite sets. For $I \subseteq M$ and $J \subseteq N$, we denote by $A_{I,J} \in \mathbb{R}^{I \times J}$ the submatrix with entries $(a_{ij})_{i \in I, j \in J}$. We abbreviate $A_{I,N}$ by $A_{I\cdot}$ and $A_{M,J}$ by $A_{\cdot J}$. For $j \in N$, we write $A_{\cdot j}$ instead of $A_{\cdot\{j\}}$ to denote the j-th column of A. Accordingly, $A_{i\cdot}$ denotes the i-th row of A, for $i \in M$. Correspondingly, for some vector $x \in \mathbb{R}^N$ and $S \subseteq N$, we denote by $x_S := (x_i)_{i \in S}$ the vector restricted to the components in S.

If $a^T x \leq \alpha$ is a valid inequality for some polyhedron P, we set $\operatorname{EQ}(P, a^T x \leq \alpha) := \{x' \in P : a^T x' = \alpha\}$. If it is clear from the context we abbreviate

$\mathrm{EQ}\,(P, a^T x \leq \alpha)$ by $\mathrm{EQ}\,(a^T x \leq \alpha)$. Each vector $x \in \mathrm{EQ}\,(a^T x \leq \alpha)$ is called a *root* of the inequality $a^T x \leq \alpha$.

A 0/1 linear program of the form $\min\{\mathbb{1}^T x : Ax \geq \mathbb{1},\, x \in \{0,1\}^N\}$ with $A \in \{0,1\}^{M \times N}$ is called a *set covering problem.* Following this definition, we call an inequality $\sum_{i \in Q} x_i \geq 1$, for some $Q \subseteq N$, a *set covering inequality.* Accordingly, if we replace in the above integer program the greater than or equal signs with less than or equal signs, i. e., we consider the problem $\max\{\mathbb{1}^T x : Ax \leq \mathbb{1}, x \in \{0,1\}^N\}$, we obtain a so-called *set packing problem.* Accordingly, we denote by $\sum_{i \in Q} x_i \leq 1$ a *set packing inequality.* Depending on the application, set packing constraints have special names, for instance, in the multiple knapsack problem (Chapter 2) these inequalities are called SOS constraints, in the Steiner tree packing case (Chapter 3) they are called *capacity constraints*, and in connection with the knapsack problem (Section 5.3) they are called *generalized upper bound constraints.*

Finally, for a subset S of a vector space, we denote by $\dim\,(S)$ the dimension of S and by $\mathrm{diff}\,(S) := \{x - y : x, y \in S\}$ the *difference set of* S.

Appendix B

Branch-and-Cut Algorithms

Branch-and-cut algorithms are currently among the most successful methods to solve integer programs and combinatorial optimization problems to optimality. In this chapter we sketch the main ideas of such an algorithm. More details and references to state-of-the-art algorithms can be found in the survey article of Caprara and Fischetti [1997]. Suppose we want to solve an integer program

$$\begin{array}{lll} \min & c^T x & \\ \text{s.t.} & Ax & \leq b, \end{array} \tag{B.1}$$

where $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$; the variables x_i $(i = 1, \dots, n)$ might be binary ($x_i \in \{0, 1\}$), integer ($x_i \in \mathbb{Z}$), or continuous ($x_i \in \mathbb{R}$). Let $P_I = \text{conv}\{x \in \mathbb{R}^n : x \text{ is feasible for (B.1)}\}$. The first step of the algorithm is to consider a relaxation of (B.1) by choosing a set $P' \subseteq \mathbb{R}^n$ with $P_I \subseteq P'$ and to optimize the linear objective function over P'. For example, this relaxation might be the linear programming relaxation $\min\{c^T x : Ax \leq b\}$ or a semidefinite relaxation. In this thesis, we only consider linear relaxations, hence, the set P' is always a polyhedron.

Let $\bar{x}$ be an optimal solution for the linear relaxation. If $\bar{x}$ is integral and all inequalities of $Ax \leq b$ are satisfied by $\bar{x}$, we have found an optimal solution for (B.1). Otherwise, there exists a hyperplane $\{x \in \mathbb{R}^n : a^T x = \alpha\}$ such that $a^T \bar{x} > \alpha$ and $P_I \subseteq \{x \in \mathbb{R}^n : a^T x \leq \alpha\}$. Such a hyperplane is called a *cutting plane*. The problem of finding such an hyperplane is called the *separation problem*. More precisely,

> Given $\bar{x} \in \mathbb{R}^n$. Decide, whether $\bar{x} \in P_I$. If not, find some valid inequality $a^T x \leq \alpha$ for P_I such that $a^T \bar{x} > \alpha$.

Sometimes, the separation problem is restricted to a certain class of inequalities (for example, to weight inequalities in the knapsack case or to alternative cycle inequalities in the Steiner tree packing case), in which case we are looking for a violated inequality of a certain class of inequalities. If we are able to find such a cutting plane, we can strengthen the relaxation and continue. This process is iterated until $\bar{x}$ is a feasible solution or no more violated inequalities are found. In the latter case this so-called *cutting plane phase* is embedded into an enumeration scheme. This is commonly done by picking some fractional variable x_i^* that must be binary or integer and creating two subproblems, one where one requires $x_i \geq \lceil x_i^* \rceil$, and one where $x_i \geq \lfloor x_i^* \rfloor$, see also the discussions in Section 5.2. The following algorithm summarizes the whole procedure.

Algorithm B.1 *Branch-and-Cut Algorithm.*

(1) Let L be a list of unsolved problems; Initialize L with (B.1).

(2) **Repeat**
(3) *Choose a problem* Π *from* L.
(4) **Repeat** (iterate)
(5) *Solve the (linear) relaxation of* Π. *Let* $\bar{x}$ *be an optimal solution.*
(6) *If* $\bar{x}$ *is feasible for* Π, Π *is solved;* **goto** *(10).*
(7) *Look for violated inequalities and add them to the LP.*
(8) **Until** *there are no violated inequalities*
(9) *Split* Π *into subproblems and add them to* L.
(10) **Until** $L = \emptyset$.
(11) *Print the optimal solution.*
(12) **STOP.**

The list L is usually organized as a binary tree, the so-called *branch-and-bound tree.* Each (sub)problem Π corresponds to a node in the tree, where the unsolved problems are the *leaves* of the tree and the node that corresponds to the entire problem (B.1) is the *root.* Branch-and-cut algorithms as outlined here are used all over the places in this thesis (see Chapters 2, 3, 4, 5, and 6). In the remainder of this chapter we discuss some implementation issues that are common to all described algorithms and that are used in basically every state-of-the-art branch-and-cut implementation.

LP-Management. The method that is commonly used to solve the LPs is the dual simplex algorithm (see also Chapter 7), because an LP basis stays dual feasible when adding cutting planes. As LP solver we use CPLEX[1], a very fast and robust linear programming solver.

Nevertheless, one major aspect in the design of a branch-and-cut algorithm is to keep the linear program of moderate size. To this end, each inequality is assigned an "age" (at the beginning the age is set to 0). Each time the inequality is not tight at the current LP solution, the age is increased by one. If the inequality gets too old, i. e., the age exceeds a certain limit, the inequality is eliminated from the LP. The value for this "age limit" varies from application to application, for example in the algorithm described in Chapter 4 this limit is set to 8, in Chapter 6 it is set to 1.

Another issue of LP-management concerns the questions: When should an inequality be added to the LP? When is an inequality considered to be "violated"? And, how many and which inequalities should be added? The answers to these questions again depend on the application. In the Steiner tree packing case, for instance, an inequality $a^T x \leq \alpha$ is considered "violated" if the slack ($= a^T \bar{x} - \alpha$) is at least 0.01. In the branch-and-cut algorithm for PIPE, see Chapter 4, we use 0.1. In `SIP` we only add inequalities if $\frac{\text{slack}}{\max a_i}$ is at least 0.05. In case too many violated inequalities are found we add in the Steiner tree packing case the 300 (300 is a parameter) most "violated" inequalities, i. e., the inequalities with the greatest slack. Another strategy is used for the solution of the matrix decomposition problem, see page 123. It is clear that we always make sure that no redundant inequalities are added to the linear program.

A commonly used structure in this context is the *pool.* Violated inequalities that are added to the LP are stored in this structure. Also inequalities that are eliminated from the LP are restored in the pool. Reasons for the pool are to reconstruct the LPs when switching from one node in the branch-and-bound tree to another and to keep inequalities that were "expensive" to separate for an easier excess in the ongoing solution process.

Heuristics. Raising the lower bound using cutting planes is one important aspect in a branch-and-cut algorithm, finding good feasible solutions early to enable fath-

[1] CPLEX is a registered trademark of ILOG

oming of branches of the search-tree is another. Primal heuristics strongly depend on the application and we have discussed these algorithm in each chapter separately. The complexity and the sensitivity to the change of the LP solutions influences the frequency in which the heuristics are called. For instance, in the PIPE case the heuristic is called only once per node in the branch-and-bound tree, whereas in the multiple knapsack case or in the Steiner tree packing case the heuristics are called after the solution of every LP.

Reduced Cost Fixing. The idea is to fix variables by exploiting the reduced costs of the current optimal LP solution. Let $\bar{z} = c^T \bar{x}$ be the objective function value of the current LP solution, z^{IP} be an upper bound on the value of the optimal solution, and $d = (d_i)_{i=1,\dots,n}$ the corresponding reduced cost vector. Consider a non-basic variable x_i of the current LP solution with finite lower and upper bounds l_i and u_i, and non-zero reduced costs $d_i > 0$. Set $\delta = \frac{z^{\mathrm{IP}} - \bar{z}}{d_i}$, rounded down in case x_j is a binary or a integer variable. Now, if x_i is currently at its lower bound l_i and $l_i + \delta < u_i$, the upper bound of x_i can be reduced to $l_i + \delta$. In case x_i is at its upper bound u_i and $u_i - \delta > l_i$, the lower bound of variable x_i can be increased to $u_i - \delta$. In case the new bounds l_i and u_i coincide, the variable can be fixed to its bounds and removed from the problem. This strengthening of the bounds is called *reduced cost fixing.* It was originally applied for binary variables (see Crowder, Johnson, and Padberg [1983]), in which case the variable can always be fixed if the criterion applies. There are problems where by the reduced cost criterion many variables can be fixed, see, for instance, the multiple knapsack problems on page 18. Sometimes, further variables can be fixed by logical implications, for example, if some binary variable x_i is fixed to one by the reduced cost criterion and it is contained in an SOS constraint, all other variables in this SOS constraint can be fixed to zero.

Enumeration Aspects. In our description of a branch-and-cut algorithm in B.1 we left the questions open which problem to choose in Step (3) and how to split the problem in Step (9). We discuss these issues in detail in Section 5.2 and refer the reader to this section for a thorough discussion.

Appendix C

Lifting

One question in polyhedral combinatorics is to extend low dimensional faces of polyhedra to higher dimensional ones. We motivate this question in the following and present the *method of sequential lifting* for solving it, see Padberg [1975], Wolsey [1975]. During the run of a branch-and-cut algorithm, see Appendix B, for solving an integer program

$$\begin{array}{rr} \min & c^T x \\ \text{s.t.} & Ax \le b \\ & 0 \le x \le u \\ & x \in \mathbb{Z}^n, \end{array} \tag{C.1}$$

where $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$, $, u \in \mathbb{Z}^n$, $b \in \mathbb{R}^m$, we must solve the separation problem: given a fractional point $\bar{x} \in \mathbb{R}^n$ that is the optimum of a linear relaxation of the integer program, find an inequality that is valid for $P_{\text{IP}} := \text{conv}\{x \in \mathbb{R}^n : Ax \le b,\, 0 \le x \le u\}$ and that cuts off this point $\bar{x}$. It is often the case that the number of fractional components of $\bar{x}$ is significantly smaller than the dimension n. To speed up the process of finding violated inequalities, one would like to restrict the search for violated inequalities to an instance of the original integer program whose dimension is equal to the number of fractional variables of $\bar{x}$. If one succeeds in finding a violated inequality in this lower dimensional space, then it remains to "lift back" the inequality of the low dimensional polyhedron to P_{IP}. One way of performing this task is to apply the method of sequential lifting that we discuss now.

Let $I \subseteq N := \{1, \ldots, n\}$ and $\sum_{i \in I} \alpha_i x_i \le \alpha_0$ be an inequality that is valid for $P_{\text{IP}} \cap \{x \in \mathbb{R}^n : x_j = 0 \text{ for all } j \in N \setminus I\}$. We assume that $\alpha_0 \in \mathbb{Z}$ and $\alpha_i \in \mathbb{Z}$ for all $i \in I$. The following algorithm extends the inequality $\sum_{i \in I} \alpha_i x_i \le \alpha_0$ to an inequality $\sum_{i \in N} \alpha_i x_i \le \alpha_0$ that is valid for P. The algorithm proceeds in an iterative fashion. It takes into account step by step a variable $i \in N \setminus I$, computes an appropriate coefficient α_i for this variable and iterates. We assume in the following that $\pi_1, \ldots, \pi_{n-|I|}$ is a permutation of the items in $N \setminus I$.

Algorithm C.2 *Sequential lifting.*

(1) **For** $k = 1$ *to* $n - |I|$ *perform the following steps:*
(2) **For** $l = 1$ *to* u_{π_k} *perform the following steps:*

$$\begin{array}{rl} \gamma(k,l) = & \max \displaystyle\sum_{i \in I} \alpha_i x_i + \sum_{i \in \{\pi_1, \ldots, \pi_{k-1}\}} \alpha_i x_i \\ & \displaystyle\sum_{i \in I} A_{\cdot i} x_i + \sum_{i \in \{\pi_1, \ldots, \pi_{k-1}\}} A_{\cdot i} x_i + A_{\cdot \pi_k} l \le b \\ & 0 \le x_i \le u_i, x_i \in \mathbb{Z} \textit{ for } i \in I \cup \{\pi_1, \ldots, \pi_{k-1}\}. \end{array} \tag{C.2}$$

(3) **End(For)**
(4) *Set*

$$\text{(C.3)} \qquad \alpha_{\pi_k} := \min_{l=1,\ldots,u_{\pi_k}} \frac{\alpha_0 - \gamma(k,l)}{l}.$$

(5) **End(For)**
(6) **Stop.**

It can be seen by induction on k that the output of this algorithm $\sum_{i\in N} \alpha_i x_i \le \alpha_0$ is a valid inequality for P_{IP}, see also Proposition 8.5.3. Note that the values α_i for $i \in N \setminus I$ are integral by our assumption on the integrality of $\sum_{i\in N} \alpha_i x_i \le \alpha_0$. In case, for some $k \in \{\pi_1, \ldots, \pi_{n-|I|}\}$, the integer program in (C.2) is infeasible, i. e., the $\gamma(k,l) = -\infty$, for all $l = 1, \ldots, u_k$, we may assign any value to α_k and the inequality stays valid. In fact, the following theorem is true, too.

Proposition C.3 *Let $I \subseteq N$ and $\sum_{i\in I} \alpha_i x_i \le \alpha_0$ an inequality that defines a facet of $P_{IP} \cap \{x \in \mathbb{R}^n : x_j = 0 \text{ for all } j \in N \setminus I\}$. After applying Algorithm C.2, the inequality $\sum_{i\in N} \alpha_i x_i \le \alpha_0$ defines a facet of P_{IP}.*

The inequality that results from applying the lifting procedure is dependent on the permutation of the items in the set $N \setminus I$.

Example C.4 *Consider the knapsack polyhedron*

$$P_{IP} = \mathrm{conv}\{x \in \{0,1\}^6 : 5x_1 + 5x_2 + 5x_3 + 5x_4 + 3x_5 + 8x_6 \le 17\}.$$

The inequality $x_1 + x_2 + x_3 + x_4 \le 3$ is valid for $P_{IP} \cap \{x_5 = x_6 = 0\}$. Choosing the permutation $(5,6)$ yields the inequality $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \le 3$. If one chooses the permutation $(6,5)$ of the items 5 and 6, the resulting inequality reads $x_1 + x_2 + x_3 + x_4 + 2x_6 \le 3$.

Note that in order to perform the lifting procedure one needs to solve a couple of integer programs that - on the first view - appear as difficult as the original problem. Sometimes they are not. For instance, if the integer program (C.1) is a 0/1 knapsack problem and the starting inequality $\sum_{i\in I} \alpha_i x_i \le \alpha_0$ is a minimal cover or $(1,k)$-configuration inequality (see page 11), the lifting coefficients can be computed in polynomial time, see Zemel [1989] or Section 5.3 for more details. It is however true that for many integer programs the lifting procedure can hardly be implemented in the way we presented it, because computing the coefficients step by step is just too expensive. In such cases, one resorts to lower bounds on the coefficients that one obtains from heuristics.

We finally note that lifting can, of course, also be applied if a variable x_i is currently at its upper bound u_i. In this case, we first "complement" variable x_i be replacing it by $u_i - x_i$, apply the same Algorithm C.2 and resubstitute the variable afterwards. Details of this procedure are discussed in Section 8.

Appendix D

Problem Data

The following tables contain statistics for the test problems considered in Part II of this thesis.

Tables D.1 and D.2 provide the data for linear programs taken from the `Netlib`. Columns 3 to 5 give the number of rows, columns and non-zeros of the problems, and Columns 6 to 8 the corresponding numbers after CPLEX presolve has been applied. These instances are available by anonymous `ftp` from `ftp://netlib2.cs.utk.edu`.

Size statistics for non-`Netlib` linear programming problems employed in our testing in Chapter 7 are given in Table D.3 in alphabetic order. The format of the table is as for the `Netlib` problems, Columns 3 to 5 give the original number of rows, columns, and non-zeros, whereas Columns 6 to 8 give these numbers after CPLEX presolve. For the most part these models were collected from proprietary models available to Bob Bixby through CPLEX Optimization, Inc. (now ILOG). With the exception of `aa6`, all models with names of the form 'aa$K$', where K is an integer, are K-variable initial segments of the 12,753,312 variable "American Airlines Challenge Model" described in Bixby, Gregory, Lustig, Marsten, and Shanno [1992].

Table D.4 contains statistics for mixed integer programming problems from the `Miplib`. Columns 2, 3, and 7 give the number of rows, columns, and non-zeros of the problems. Columns 3 to 5 provide the number of binary, integer, and continuous variables. A description of the problems and the data are available from `URL` `http://www.caam.rice.edu:80/~bixby/miplib/miplib.html`, see Bixby, Ceria, McZeal, and Savelsbergh [1998].

Example	Original			Presolved		
	Rows	Cols	NZs	Rows	Cols	NZs
25fv47	821	1571	10400	684	1449	9903
80bau3b	2262	9799	21002	1965	8680	18981
adlittle	56	97	383	53	94	372
afiro	27	32	83	20	28	71
agg	488	163	2410	164	107	867
agg2	516	302	4284	280	250	2267
agg3	516	302	4300	282	249	2298
bandm	305	472	2494	179	228	1531
beaconfd	173	262	3375	49	105	1033
blend	74	83	491	51	57	394
bnl1	643	1175	5121	451	995	4632
bnl2	2324	3489	13999	943	2095	10252
boeing1	351	384	3485	287	419	2765
boeing2	166	143	1196	122	160	811
bore3d	233	315	1429	52	74	411
brandy	220	249	2148	108	177	1667
capri	271	353	1767	159	224	1304
cycle	1903	2857	20720	929	1791	12993
czprob	929	3523	10669	464	2491	4982
d2q06c	2171	5167	32417	1875	4617	30600
degen2	444	534	3978	382	473	3851
degen3	1503	1818	24646	1407	1722	24427
dfl001	6071	12230	35632	3965	9212	32153
e226	223	282	2578	148	251	2267
etamacro	400	688	2409	294	478	1910
fffff800	524	854	6227	295	638	4804
finnis	497	614	2310	340	404	1426
fit1d	24	1026	13404	24	1024	13386
fit1p	627	1677	9868	627	1427	9618
fit2d	25	10500	129018	25	10450	128564
fit2p	3000	13525	50284	3000	13525	50284
forplan	161	421	4563	101	364	3801
ganges	1309	1681	6912	576	803	4187
gfrdpnc	616	1092	2377	322	794	1781
greenbea	2392	5405	30877	1020	3058	23028
greenbeb	2392	5405	30877	1019	3049	22927
grow15	300	645	5620	300	645	5620
grow22	440	946	8252	440	946	8252
grow7	140	301	2612	140	301	2612
israel	174	142	2269	163	141	2256
kb2	43	41	286	39	32	266
lotfi	153	308	1078	117	282	596
maros	846	1443	9614	539	843	5788
nesm	662	2923	13288	622	2707	12933
perold	625	1376	6018	507	1096	5359

Table D.1: Problem statistics for `Netlib` LP problems.

Example	Original			Presolved		
	Rows	Cols	NZs	Rows	Cols	NZs
pilot4	410	1000	5141	353	773	4705
pilot87	2030	4883	73152	1890	4511	70370
pilotja	940	1988	14698	745	1420	10985
pilotnov	975	2172	13057	785	1737	11528
pilots	1441	3652	43167	1275	3243	40467
pilotwe	722	2789	9126	624	2378	8311
recipe	91	180	663	55	89	395
sc105	105	103	280	59	58	266
sc205	205	203	551	116	115	611
sc50a	50	48	130	29	28	96
sc50b	50	48	118	28	28	84
scagr25	471	500	1554	240	391	1223
scagr7	129	140	420	60	103	305
scfxm1	330	457	2589	237	383	2148
scfxm2	660	914	5183	476	768	4321
scfxm3	990	1371	7777	715	1153	6494
scorpion	388	358	1426	102	140	532
scrs8	490	1169	3182	158	809	2514
scsd1	77	760	2388	77	760	2388
scsd6	147	1350	4316	147	1350	4316
scsd8	397	2750	8584	397	2750	8584
sctap1	300	480	1692	269	339	1444
sctap2	1090	1880	6714	977	1326	5717
sctap3	1480	2480	8874	1344	1767	7630
seba	515	1028	4352	2	8	11
share1b	117	225	1151	103	204	1048
share2b	96	79	694	93	79	691
shell	536	1775	3556	248	1204	2414
ship04l	402	2118	6332	288	1886	4267
ship04s	402	1458	4352	188	1238	2804
ship08l	778	4283	12802	470	3099	7100
ship08s	778	2387	7114	234	1538	3534
ship12l	1151	5427	16170	609	4147	9222
ship12s	1151	2763	8178	267	1847	4121
sierra	1227	2036	7302	1094	1916	6966
stair	356	467	3856	242	270	3520
standata	359	1183	3031	250	717	1600
standmps	467	1075	3679	352	969	2344
stocfor1	117	111	447	61	63	349
stocfor2	2157	2031	8343	1362	1248	7022
stocfor3	16675	15695	64875	10740	9786	52492
truss	1000	8806	27836	1000	8806	27836
tuff	333	587	4520	142	388	4041
vtpbase	198	203	908	49	78	227
wood1p	244	2594	70215	170	1728	44884
woodw	1098	8405	37474	555	4010	14536

Table D.2: Problem statistics for `Netlib` LP problems.

Example	Original			Presolved		
	Rows	Cols	NZs	Rows	Cols	NZs
0321.4	1202	71201	818258	1202	50559	656073
0341.4	658	46508	384286	658	27267	264239
aa100000	837	100000	770645	837	68428	544654
aa1000000	837	1000000	7887318	837	604371	5051196
aa200000	837	200000	1535412	837	134556	1075761
aa25000	837	25000	192313	837	17937	140044
aa300000	837	300000	2314117	837	197764	1595300
aa400000	837	400000	3115729	837	259924	2126937
aa50000	837	50000	380535	837	35331	276038
aa500000	837	500000	3889641	837	320228	2624731
aa6	541	4486	25445	532	4316	24553
aa600000	837	600000	4707661	837	378983	3138105
aa6000000	837	6000000	46972327	837	2806468	23966705
aa700000	837	700000	5525946	837	434352	3620867
aa75000	837	75000	576229	837	52544	415820
aa800000	837	800000	6309846	837	493476	4112683
aa900000	837	900000	7089709	837	548681	4575788
amax	5160	150000	6735560	5084	150000	3237088
continent	10377	57253	198214	6841	45771	158025
cre_b	9648	72447	256095	5229	31723	107169
finland	56794	139121	658616	5372	61505	249100
fit2d	25	10500	129018	25	10450	128564
food	27349	97710	288421	10544	69004	216325
imp1	4089	121871	602491	1587	112201	577607
mctaq	1129	16336	52692	1129	16336	52692
nopert	1119	16336	50749	1119	16336	50749
nw16	139	148633	1501820	139	138951	1397070
osa030	4350	100024	600144	4279	96119	262872
osa060	10280	232966	1397796	10209	224125	584253
pilots	1441	3652	43167	1275	3243	40467
ra1	823	8904	72965	780	8902	70181
roadnet	463	42183	394187	462	41178	383857
sfsu2	4246	55293	984777	3196	53428	783198
sfsu3	1973	60859	2111658	1873	60716	2056445
sfsu4	2217	33148	437095	1368	24457	180067
tm	28420	164024	505253	17379	139529	354697
us01	145	1053137	13636541	87	370626	3333071
usfs2	1484	13822	158612	1166	12260	132531
w1.dual	42	415953	3526288	22	140433	1223824

Table D.3: Problem statistics for non-`Netlib` LP problems.

Example	Rows	Cols	Bin	Int	Cont	NZs
10teams	230	2025	1800	0	225	12150
air03	124	10757	10757	0	0	91028
air04	823	8904	8904	0	0	72965
air05	426	7195	7195	0	0	52121
arki001	1048	1388	415	123	850	20439
bell3a	123	133	39	32	62	347
bell5	91	104	30	28	46	266
blend2	274	353	231	33	89	1409
cap6000	2176	6000	6000	0	0	48249
dano3mip	3202	13873	552	0	13321	79655
danoint	664	521	56	0	465	3232
dcmulti	290	548	75	0	473	1315
dsbmip	1182	1886	160	32	1694	7366
egout	98	141	55	0	86	282
enigma	21	100	100	0	0	289
fast0507	507	63009	63009	0	0	409349
fiber	363	1298	1254	0	44	2944
fixnet6	478	878	378	0	500	1756
flugpl	18	18	0	11	7	46
gen	780	870	144	6	720	2592
gesa2	1392	1224	240	168	816	5064
gesa2_o	1248	1224	384	336	504	3672
gesa3	1368	1152	216	168	768	4944
gesa3_o	1224	1152	336	336	480	3624
gt2	29	188	24	164	0	376
harp2	112	2993	2993	0	0	5840
khb05250	101	1350	24	0	1326	2700
l152lav	97	1989	1989	0	0	9922
lseu	28	89	89	0	0	309
misc03	96	160	159	0	1	2053
misc06	820	1808	112	0	1696	5859
misc07	212	260	259	0	1	8619
mitre	2054	10724	10724	0	0	39704
mod008	6	319	319	0	0	1243
mod010	146	2655	2655	0	0	11203
mod011	4480	10958	96	0	10862	22254
modglob	291	422	98	0	324	968
noswot	182	128	75	25	28	735
nw04	36	87482	87482	0	0	636666
p0033	16	33	33	0	0	98
p0201	133	201	201	0	0	1923
p0282	241	282	282	0	0	1966
p0548	176	548	548	0	0	1711
p2756	755	2756	2756	0	0	8937
pk1	45	86	55	0	31	915
pp08a	136	240	64	0	176	480
pp08aCUTS	246	240	64	0	176	839
qiu	1192	840	48	0	792	3432
qnet1	503	1541	1288	129	124	4622
qnet1_o	456	1541	1288	129	124	4214
rentacar	6803	9557	55	0	9502	41868
rgn	24	180	100	0	80	460
rout	291	556	300	15	241	2431
set1ch	492	712	240	0	472	1412
seymour	4944	1372	1372	0	0	33549
stein27	118	27	27	0	0	378
stein45	331	45	45	0	0	1034
vpm1	234	378	168	0	210	749
vpm2	234	378	168	0	210	917

Table D.4: Problem statistics for `Miplib`-problems

Bibliography

Aardal, K. and Weismantel, R. (1997). *Polyhedral Combinatorics.* In Dell'Amico, Maffioli, and Martello [1997], chapter 3, pages 31–44.

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

Alevras, D., Grötschel, M., and Wessäly, R. (1998). Cost–efficient network synthesis from leased lines. *Annals of Operations Research*, 76:1–20.

Andersen, E. D. and Andersen, K. D. (1995). Presolving in linear programming. *Mathematical Programming*, 71:221 – 245.

Applegate, D., Bixby, R. E., Chvátal, V., and Cook, W. (1995). Finding cuts in the TSP. Technical Report 95-05, DIMACS.

Applegate, D., Bixby, R. E., Chvátal, V., and Cook, W. (1998). Project-and-lift (a paradigm for finding cuts). Draft.

Balas, E. (1975). Facets of the knapsack polytope. *Mathematical Programming*, 8:146 – 164.

Balas, E., Ceria, S., and Cornuéjols, G. (1993). A lift-and-project cutting plane algorithm for mixed $0-1$ programs. *Mathematical Programming*, 58:295–324.

Balas, E., Ceria, S., and Cornuéjols, G. (1996). Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42:1229 – 1246.

Balas, E., Ceria, S., Cornuéjols, G., and Natraj, N. (1996). Gomory cuts revisited. *Operations Research Letters*, 19:1 – 9.

Balas, E. and Ho, A. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study. *Mathematical Programming*, 12:37–60.

Barr, R. S. and Hickman, B. L. (1994). Parallel simplex for large pure network problems: Computational testing and sources of speedup. *Operations Research*, 42:65 – 80.

Bauer, B. E. (1992). *Practical Parallel Programming.* Academic Press.

Beguelin, A., Dongarra, J., Geist, A., Jiang, W., Manchek, R., and Sunderam, V. (1994). *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing.* The MIT Press, Cambride, Massachusetts, London, England. Information available via WWW at URL http://www.epm.ornl.gov/pvm/pvm_home.html.

Benders, J. F. (1962). Partitioning procedures for solving mixed variables programming. *Numerische Mathematik*, 4:238–252.

Benichou, M., Gauthier, J. M., Girodet, P., Hentges, G., Ribiere, G., and Vincent, O. (1971). Experiments in mixed-integer programming. *Mathematical Programming*, 1:76 – 94.

Bienstock, D., Chopra, S., Günlük, O., and Tsai, C.-Y. (1998). Minimum cost capacity installation for multicommodity network flows. *Mathematical Programming*, 81:177 – 199.

Bienstock, D. and Günlük, O. (1996). Capacitated network design – Polyhedral structure and computation. *INFORMS Journal on Computing*, 8:243–259.

Bixby, R. E. (1994). *Lectures on Linear Programming*. Rice University, Houston, Texas.

Bixby, R. E., Ceria, S., McZeal, C., and Savelsbergh, M. W. P. (1998). An updated mixed integer programming library: MIPLIB 3.0. Paper and Problems available at WWW Page: http://www.caam.rice.edu/~bixby/miplib/miplib.html.

Bixby, R. E., Gregory, J. W., Lustig, I. J., Marsten, R. E., and Shanno, D. F. (1992). Very large-scale linear programming: A case study in combining interior point and simplex methods. *Operations Research*, 40:885 – 897.

Bixby, R. E. and Martin, A. (1995). Parallelizing the dual simplex method. Preprint SC 95-45, Konrad-Zuse-Zentrum Berlin.

Bondy, J. A. and Murty, U. S. R. (1976). *Graph Theory with Applications*. American Elsevier, New York, and Macmillan, London.

Borndörfer, R. (1998). *Aspects of Set Packing, Partitioning, and Covering*. PhD thesis, Technische Universität Berlin.

Borndörfer, R., Ferreira, C. E., and Martin, A. (1998). Decomposing matrices into blocks. *SIAM Journal on Optimization*, 9:236 – 269.

Borndörfer, R. and Weismantel, R. (1997a). Relations among some combinatorial programs. Technical Report Preprint SC 97-54, Konrad-Zuse-Zentrum für Informationstechnik Berlin.

Borndörfer, R. and Weismantel, R. (1997b). Set packing relaxations of some integer programs. Technical Report Preprint SC 97-30, Konrad-Zuse-Zentrum für Informationstechnik Berlin.

Brady, M. L. and Brown, D. J. (1984). VLSI routing: Four layers suffice. In Preparata, F. P., editor, *VLSI theory*, volume 2 of *Advances in Computing Research*, pages 245 – 258. Jai Press, London.

Burstein, M. and Pelavin, R. (1983). Hierarchical wire routing. *IEEE Transactions on Computer-Aided-Design*, 2:223 – 234.

Caprara, A. and Fischetti, M. (1997). Branch-and-cut algorithms. In Dell'Amico, M., Maffioli, F., and Martello, S., editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 45–63. John Wiley & Sons Ltd, Chichester.

Carøe, C. C., Ruszczyński, A., and Schultz, R. (1997). Unit commitment under uncertainty via two-stage stochastic programming. Technical Report DIKU-TR-97123, Department of Computer Science, University of Copenhagen.

Ceria, S., Cordier, C., Marchand, H., and Wolsey, L. A. (1998). Cutting planes for integer programs with general integer variables. *Mathematical Programming*, 81:201 – 214.

Chang, M. D., Engquist, M., Finkel, R., and Meyer, R. R. (1988). A parallel algorithm for generalized networks. *Annals of Operations Research*, 14:125 – 145.

Chartrand, G. and Lesniak, L. (1986). *Graphs & digraphs (Second edition)*. Wadsworth & Brooks/Cole Advanced Books and Software, Pacific Grove, California, USA.

Chvátal, V. (1973). Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305 – 337.

Chvátal, V. (1983). *Linear Programming.* W. H. Freeman and Company.

Cohoon, J. P. and Heck, P. L. (1988). BEAVER: A computational-geometry-based tool for switchbox routing. *IEEE Transactions on Computer-Aided-Design*, 7:684 – 697.

Cordier, C., Marchand, H., Laundy, R., and Wolsey, L. A. (1997). bc – opt: a branch-and-cut code for mixed integer programs. Technical Report CORE DP9778, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

CPLEX (1997). *Using the CPLEX Callable Library.* ILOG CPLEX Division, 889 Alder Avenue, Suite 200, Incline Village, NV 89451, USA. Information available at URL http://www.cplex.com.

Crama, Y. and Oosten, M. (1996). Models for machine-part grouping in cellular manufacturing. *Int. J. Prod. Res.*, 34:1693 – 1713.

Crowder, H., Johnson, E., and Padberg, M. W. (1983). Solving large-scale zero-one linear programming problems. *Operations Research*, 31:803–834.

Dahl, G., Martin, A., and Stoer, M. (1995). Routing through virtual paths in layered telecommunication networks. Research Note N 78/95, Telenor Research and Development, Kjeller, Norway. To appear in Operations Research.

Dantzig, G. B. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8:101–111.

Davis, T. A. and Yewr, P. (1990). A nondeterministic parallel algorithm for general unsymmetric sparse LU factorization. *SIAM Journal on Matrix Analysis and Application*, 11:383 – 402.

Dell'Amico, M., Maffioli, F., and Martello, S., editors (1997). *Annotated Bibliographies in Combinatorial Optimization.* John Wiley & Sons Ltd, Chichester.

Dreyfus, S. E. and Wagner, R. A. (1971). The Steiner problem in graphs. *Networks*, 1:195 – 207.

Duff, I. S., Erisman, A. M., and Reid, J. K. (1986). *Direct Methods for Sparse Matrices.* Oxford University Press.

Eckstein, J. (1994). Parallel branch-and-bound algorithms for general mixed integer programming on the CM-5. *SIAM Journal on Optimization*, 4:794 – 814.

Eckstein, J., Boduroglu, I. I., Polymenakos, L. C., and Goldfarb, D. (1995). Data-parallel implementation of dense simplex methods on the connection machine CM-2. *ORSA Journal on Computing*, 7:402 – 416.

Ehrgott, M. (1992). Optimierungsprobleme in Graphen unter Kardinalitätsrestriktionen. Master's thesis, Univ. Kaiserslautern, Dept. of Mathematics.

Erickson, R. E., Monma, C. L., and Veinott, A. F. (1987). Send-and-split method for minimum concave-cost network flows. *Mathematics of Operations Research*, 12:634 – 664.

Ferreira, C. E. (1994). *On Combinatorial Optimization Problems Arising in Computer System Design.* PhD thesis, Technische Universität Berlin.

Ferreira, C. E., Grötschel, M., Kiefl, S., Krispenz, C., Martin, A., and Weismantel, R. (1993). Some integer programs arising in the design of main frame computers. *ZOR – Methods and Models of Operations Research*, 38:77 – 100.

Ferreira, C. E., Martin, A., de Souza, C. C., Weismantel, R., and Wolsey, L. A. (1998). The node capacitated graph partitioning problem: A computational study. *Mathematical Programming*, 81:229 – 256.

Ferreira, C. E., Martin, A., and Weismantel, R. (1996). Solving multiple knapsack problems by cutting planes. *SIAM Journal on Optimization*, 6:858 – 877.

Ferris, M. C. and Horn, J. D. (1998). Partitioning mathematical programs for parallel solution. *Mathematical Programming*, 80:35 – 61.

Fiduccia, C. M. and Mattheyses, R. M. (1982). A linear-time heuristic for improving network partitions. *Proc. 19th. DAC*, pages 175–181.

Forrest, J. J. H. and Goldfarb, D. (1992). Steepest-edge simplex algorithms for linear programming. *Mathematical Programming*, 57:341 – 374.

Forrest, J. J. H. and Tomlin, J. A. (1972). Updating triangular factors of the basis to maintain sparsity in the product-form simplex method. *Mathematical Programming*, 2:263 – 278.

Frank, A. (1990). Packing paths, circuits, and cuts – a survey. In Korte, B., Lovász, L., Prömel, H. J., and Schrijver, A., editors, *Paths, Flows, and VLSI-Layout*, pages 47 – 100. Springer, Berlin Heidelberg.

Gallivan, K. A., Heath, M. T., Ng, E., Ortega, J. M., Peyton, B. W., Plemmons, R. J., Romine, C. H., Sameh, A. H., and Voigt, R. G. (1990). *Parallel Algorithms for Matrix Computations*. Society for Industrial and Applied Mathematics, Philadelphia.

Garey, M. R. and Johnson, D. S. (1977). The rectilinear Steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32:826 – 834.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

Gerez, S. H. and Herrmann, O. E. (1989). Switchbox routing by stepwise reshaping. *IEEE Transactions on Computer-Aided-Design*, 8:1350 – 1361.

Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H. (1989). A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45:437 – 474.

Goldberg, A. V. and Tarjan, R. E. (1988). A new approach to the maximum flow problem. *Journal of ACM*, 35:921 – 940.

Gomory, R. E. (1960). Solving linear programming problems in integers. In Bellman, R. and Hall, M., editors, *Combinatorial analysis, Proceedings of Symposia in Applied Mathematics*, volume 10, Providence RI.

Gomory, R. E. (1969). An algorithm for integer solutions to linear programming. In Graves, R. L. and Wolfe, P., editors, *Recent Advances in Mathematical Programming*, pages 269 – 302, New York. McGraw-Hill.

Gondzio, J. (1994). Presolve analysis of linear programs prior to apply an interior point method. Technical Report 1994.3, University of Geneva, Switzerland.

Gottlieb, E. S. and Rao, M. R. (1990a). $(1,k)$-configuration facets for the generalized assignment problem. *Mathematical Programming*, 46:53–60.

Gottlieb, E. S. and Rao, M. R. (1990b). The generalized assignment problem: Valid inequalities and facets. *Mathematical Programming*, 46:31–52.

Grötschel, M., Lovász, L., and Schrijver, A. (1988). *Geometric Algorithms and Combinatorial Optimization*. Springer.

Grötschel, M., Martin, A., and Weismantel, R. (1995). Routing in grid graphs by cutting planes (extended version). *ZOR – Methods and Models of Operations Research*, 41:255 – 275.

Grötschel, M., Martin, A., and Weismantel, R. (1996b). Packing Steiner trees: A cutting plane algorithm and computational results. *Mathematical Programming*, 72:125 – 145.

Grötschel, M., Martin, A., and Weismantel, R. (1996c). Packing Steiner trees: Further facets. *European Journal on Combinatorics*, 17:39 – 52.

Grötschel, M., Martin, A., and Weismantel, R. (1996a). Packing Steiner trees: Polyhedral investigations. *Mathematical Programming*, 72:101 – 123.

Grötschel, M., Martin, A., and Weismantel, R. (1996d). Packing Steiner trees: Separation algorithms. *SIAM Journal on Discrete Mathematics*, 9:233 – 257.

Grötschel, M., Martin, A., and Weismantel, R. (1997). The Steiner tree packing problem in VLSI-design. *Mathematical Programming*, 78:265 – 281.

Grötschel, M. and Monma, C. L. (1990). Integer polyhedra associated with certain network design problems with connectivity constraints. *SIAM Journal on Discrete Mathematics*, 3:502 – 523.

Grötschel, M., Monma, C. L., and Stoer, M. (1992). Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations Research*, 40:309 – 330.

Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W. P. (1994). Lifted cover inequalities for $0-1$ integer programs: computation. Technical Report LEC 94-09, Georgia Institute of Technology.

Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W. P. (1997). Lifted cover inequalities for $0-1$ integer programs: Complexity. Technical report, Georgia Institute of Technology. to appear in INFORMS Journal on Computing.

Gupta, A. (1996). Fast and effective algorithms for graph partitioning and sparse matrix ordering. Research Report RC 20496, IBM T. J. Watson Research Center, Yorktown Heights.

Hammer, P. L., Johnson, E. L., and Peled, U. N. (1975). Facets of regular 0-1 polytopes. *Mathematical Programming*, 8:179 – 206.

Harris, P. M. J. (1973). Pivot selection methods of the devex LP code. *Mathematical Programming*, 5:1 – 28.

Helgason, R. V., Kennington, J. L., and Zaki, H. A. A. (1988). Parallelization of the simplex method. *Annals of Operations Research*, 14:17 – 40.

Helmberg, C., Mohar, B., Poljak, S., and Rendl, F. (1995). A spectral approach to bandwidth and separator problems in graphs. *Linear and Multilinear Algebra*, 39:73–90.

Hoffman, K. L. and Padberg, M. W. (1991). Improved LP-representations of zero-one linear programs for branch-and-cut. *ORSA Journal on Computing*, 3:121–134.

Hoffman, K. L. and Padberg, M. W. (1993). Solving airline crew-scheduling problems by branch-and-cut. *Management Science*, 39:657–682.

Johnson, E. and Padberg, M. W. (1981). A note on the knapsack problem with special ordered sets. *Operations Research Letters*, 1:18 – 22.

Joobbani, R. and Siewiorek, D. P. (1986). WEAVER: A knowledge-based routing expert. *IEEE Design and Test*, pages 12 – 23.

Jou, J. M., Lee, J. Y., Sun, Y., and Wang, J. F. (1990). An efficient VLSI switch-box router. *IEEE Design and Test*, pages 52 – 65.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations*, pages 85 – 103. Plenum Press, New York.

Klabjan, D., Nemhauser, G. L., and Tovey, C. (1996). The complexity of cover inequality separation. Technical report, Georgia Institute of Technology, Atlanta, USA. To appear in Operations Research Letters.

Koch, T. and Martin, A. (1998). Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207 – 232.

Korte, B., Prömel, H. J., and Steger, A. (1990). Steiner trees in VLSI-layout. In Korte, B., Lovász, L., Prömel, H. J., and Schrijver, A., editors, *Paths, Flows, and VLSI-Layout*, pages 329 – 371. Springer, Berlin Heidelberg.

Kramer, M. R. and van Leeuwen, J. (1984). The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits. In Preparata, F. P., editor, *VLSI theory*, volume 2 of *Advances in Computing Research*, pages 129 – 146. Jai Press, London.

Kumar, V., Grama, A., Gupta, A., and Karypis, G. (1994). *Introduction to Parallel Computing*. The Benjamin/Cummings Publishing Company, Inc.

Land, A. and Powell, S. (1979). Computer codes for problems of integer programming. *Annals of Discrete Mathematics*, 5:221 – 269.

Laurent, M. and Sassano, A. (1992). A characterization of knapsacks with the max-flow-min-cut property. *Operations Research Letters*, 11:105–110.

Lengauer, T. (1990). *Combinatorial algorithms for integrated circuit layout*. Wiley, Chichester.

Lin, Y. L., Hsu, Y. C., and Tsai, F. S. (1988). A detailed router based on simulated evolution. *Proc. Int. Conf. Computer-Aided-Design*, pages 38 – 41.

Linderoth, J. T. and Savelsbergh, M. W. P. (1997). A computational study of search strategies for mixed integer programming. Technical Report LEC-97-12, Georgia Institute of Technology.

Lipski, W. (1984). On the structure of three-layer wireable layouts. In Preparata, F. P., editor, *VLSI theory*, volume 2 of *Advances in Computing Research*, pages 231 – 244. Jai Press, London.

Löbel, A. (1997). *Optimal Vehicle Scheduling in Public Transit*. PhD thesis, Technische Universität Berlin.

Lomonosov, M. V. (1985). Combinatorial approaches to multiflow problems. *Discrete Applied Mathematics*, 11:1 – 94.

Lorentzen, R. (1994). Mathematical methods and algorithms in the network utilization planning tool ruginett. In *Telektronikk*, volume 90, pages 73 – 82, Telenor Research, P.O.Box 83, 2007 Kjeller, Norway.

Lovász, L. and Plummer, M. D. (1986). *Matching theory*. North-Holland.

Luk, W. K. (1985). A greedy switch-box router. *Integration*, 3:129 – 149.

Magnanti, T. L., Mirchandani, P., and Vachani, R. (1995). Modeling and solving the two–facility capacitated network loading problem. *Operations Research*, 43:142–157.

Marchand, H. (1998). *A Polhedral Study of the Mixed Knapsack Set and its Use to Solve Mixed Integer Programs*. PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

Marchand, H. and Wolsey, L. A. (1997). The $0-1$ knapsack problem with a single continuous variable. Technical Report CORE DP9720, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

Martello, S. and Toth, P. (1990). *Knapsack Problems. Algorithms and computer implementations.* John Wiley & Sons Ltd, Chichester.

Martin, A. (1992). *Packen von Steinerbäumen: Polyedrische Studien und Anwendungen.* PhD thesis, Technische Universität Berlin.

Martin, A. and Weismantel, R. (1997). Contributions to general mixed integer knapsack problems. Preprint SC 97-38, Konrad-Zuse-Zentrum Berlin.

Martin, A. and Weismantel, R. (1998). The intersection of knapsack polyhedra and extensions. In Bixby, R. E., Boyd, E. A., and Ríos-Mercado, R. Z., editors, *Integer Programming and Combinatorial Optimization*, Proceedings of the 6th IPCO Conference, pages 243 – 256.

Mitra, G. (1973). Investigations of some branch and bound strategies for the solution of mixed integer linear programs. *Mathematical Programming*, 4:155 – 170.

Nemhauser, G. L., Savelsbergh, M. W. P., and Sigismondi, G. C. (1994). MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47 – 58.

Nemhauser, G. L. and Vance, P. H. (1994). Lifted cover facets of the $0-1$ knapsack polytope with GUB constraints. *Operations Research Letters*, 16:255 – 263.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization.* Wiley.

Nemhauser, G. L. and Wolsey, L. A. (1990). A recursive procedure to generate all cuts for $0-1$ mixed integer programs. *Mathematical Programming*, 46:379 – 390.

Nicoloso, S. and Nobili, P. (1992). A set covering formulation of the matrix equipartition problem. In Kall, P., editor, *System Modelling and Optimization, Proceedings of the 15th IFIP conference, Zürich, September 1991*, pages 189 – 198. Springer.

Nobili, P. and Sassano, A. (1989). Facets and lifting procedures for the set covering polytope. *Mathematical Programming*, 45:111–137.

Nobili, P. and Sassano, A. (1992). A separation routine for the set covering polytope. In Balas, E., Cornuéjols, G., and Kannan, R., editors, *Integer Programming and Combinatorial Optimization*, Proceedings of the 2nd IPCO Conference, pages 201 – 219.

Padberg, M. W. (1973). On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215.

Padberg, M. W. (1975). A note on zero-one programming. *Operations Research*, 23:833–837.

Padberg, M. W. (1980). $(1,k)$-configurations and facets for packing problems. *Mathematical Programming*, 18:94–99.

Padberg, M. W. (1995). *Linear Optimization and Extensions.* Springer.

Padberg, M. W., Roy, T. J. V., and Wolsey, L. A. (1985). Valid inequalities for fixed charge problems. *Operations Research*, 33:842 – 861.

Park, K., Kang, S., and Park, S. (1994). An integer programming approach to the bandwidth packing problem. Technical report, Dept. of Industrial Engineering, Korea Advanced Institute of Science and Technology, Taejon, Korea.

Parker, M. and Ryan, J. (1994). A column generation algorithm for bandwidth packing. *Telecommunications Systems*, 2:185 – 195.

Peters, J. (1990). The network simplex method on a multiprocessor. *Networks*, 20:845 – 859.

Pothen, A., Simon, H. D., and Liou, K.-P. (1990). Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis*, 11:430–452.

Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389 – 1401.

Robertson, N. and Seymour, P. D. (1990). An outline of a disjoint paths algorithm. In Korte, B., Lovász, L., Prömel, H. J., and Schrijver, A., editors, *Paths, Flows, and VLSI-Layout*, pages 267 – 292. Springer, Berlin Heidelberg.

Rothberg, E. and Hendrickson, B. (1996). Sparse matrix ordering methods for interior point linear programming. Technical Report SAND96-0475J, Sandia National Laboratories.

Roy, T. J. V. and Wolsey, L. A. (1987). Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35:45 – 57.

Sarrafzadeh, M. (1987). Channel-routing problem in the knock-knee mode is NP-complete. *IEEE Transactions on Computer-Aided-Design*, 6:503 – 506.

Schrijver, A. (1980). On cutting planes. *Annals of Discrete Mathematics*, 9:291 – 296.

Schrijver, A. (1986). *Theory of Linear and Integer Programming*. Wiley, Chichester.

Schrijver, A. (1990). Homotopic routing methods. In Korte, B., Lovász, L., Prömel, H. J., and Schrijver, A., editors, *Paths, Flows, and VLSI-Layout*, pages 329 – 371. Springer, Berlin Heidelberg.

Sharda, R. (1995). Linear programming solver software for personal computers: 1995 report. *OR/MS Today*, 22(5):49 – 57.

Soumis, F. (1997). *Decomposition and Column Generation*. In Dell'Amico, Maffioli, and Martello [1997], chapter 8, pages 115 – 126.

Stevens, W. R. (1990). *Unix Network Programming*. Prentice-Hall.

Stoer, M. and Dahl, G. (1994). A polyhedral approach to multicommodity survivable network design. *Numerische Mathematik*, 68:149–167.

Stougie, L. and van der Vlerk, M. H. (1997). *Stochastic Integer Programming*. In Dell'Amico, Maffioli, and Martello [1997], chapter 9, pages 127 – 141.

Suhl, U. H. and Szymanski, R. (1994). Supernode processing of mixed-integer models. *Computational Optimization and Applications*, 3:317 – 331.

Szymanski, T. G. (1985). Dogleg channel routing is NP-complete. *IEEE Transactions on Computer-Aided-Design*, 4:31 – 40.

Thienel, S. (1995). *ABACUS A Branch-And-CUt System*. PhD thesis, Universität zu Köln.

Tomlin, J. A. and Welsh, J. S. (1986). Finding duplicate rows in a linear program. *Operations Research Letters*, 5:7 – 11.

Truemper, K. (1992). *Matroid Decomposition*. Acadamic Press.

Truemper, K. (1997). Personal communication.

Tzeng, P. and Séquin, C. H. (1988). Codar: a congestion-directed general area router. *Proc. Int. Conf. Computer-Aided Design*, pages 30 – 33.

Weismantel, R. (1997). On the 0/1 knapsack polytope. *Mathematical Programming*, 77:49–68.

Wolsey, L. A. (1975). Faces of linear inequalities in 0-1 variables. *Mathematical Programming*, 8:165 – 178.

Wolsey, L. A. (1990). Valid inequalities for 0-1 knapsacks and MIPs with generalized upper bound constraints. *Discrete Applied Mathematics*, 29:251–261.

Wunderling, R. (1996). Paralleler und objektorientierter Simplex. Technical Report TR 96-09, Konrad-Zuse-Zentrum Berlin.

Zemel, E. (1989). Easily computable facets of the knapsack polytope. *Mathematics of Operations Research*, 14:760 – 764.

Index