K. Aardal et al., Eds., *Handbooks in OR & MS, Vol. 12*

Chapter 2

Computational Integer Programming and Cutting Planes

Armin Fügenschuh and Alexander Martin

Abstract

The study and solution of mixed-integer programming problems is of great interest, because they arise in a variety of mathematical and practical applications. Today's state-of-art software packages for solving mixed-integer programs based on linear programming include preprocessing, branch-and-bound, and cutting planes techniques. The main purpose of this article is to describe these components and recent developments that can be found in many solvers. Besides linear programming based relaxation methods we also discuss Langrangean, Dantzig–Wolfe and Benders' decomposition and their interrelations.

1 Introduction

The study and solution of linear mixed integer programs lies at the heart of discrete optimization. Various problems in science, technology, business, and society can be modeled as linear mixed integer programming problems and their number is tremendous and still increasing. This handbook, for instance, documents the variety of ideas, approaches and methods that help to solve mixed integer programs, since there is no unique method that solves them all, see also the surveys Aardal, Weismantel, and Wolsey (2002); Johnson, Nemhauser, and Savelsbergh (2000); Marchand, Martin, Weismantel, and Wolsey (2002). Among the currently most successful methods are linear programming (LP, for short) based branch-and-bound algorithms where the underlying linear programs are possibly strengthened by cutting planes. For example, most commercial mixed integer programming solvers, see Sharda (1995), or special purpose codes for problems like the traveling salesman problem are based on this method.

The purpose of this chapter is to describe the main ingredients of today's (commercial or research oriented) solvers for integer programs. We assume the reader to be familiar with basics in linear programming and polyhedral theory, see for instance Chvátal (1983) or Padberg (1995).

Consider an integer program or more general a mixed integer program (MIP) in the form

$$\begin{aligned} z_{\mathrm{MIP}} = \min \quad & c^T x \\ \text{s.t.} \quad & Ax \left\{ \begin{matrix} \leq \\ = \end{matrix} \right\} b \\ & l \leq x \leq u \\ & x \in \mathbb{Z}^N \times \mathbb{R}^C, \end{aligned} \tag{1}$$

where $A \in \mathbb{Q}^{M \times (N \cup C)}$, $c \in \mathbb{Q}^{N \cup C}$, $b \in \mathbb{Q}^M$. Here, M, N and C are nonempty, finite, ordered sets with N and C disjoint. Without loss of generality, we may assume that the elements of N and C are represented by numbers, i.e., $N = \{1, \ldots, p\}$ and $C = \{p+1, \ldots, n\}$. The vectors $l \in (\mathbb{Q} \cup \{-\infty\})^{N \cup C}$, $u \in (\mathbb{Q} \cup \{\infty\})^{N \cup C}$ are called *lower* and *upper bounds* on x, respectively. A variable x_j, $j \in N \cup C$, is *unbounded from below* (*above*), if $l_j = -\infty$ ($u_j = \infty$). An integer variable $x_j \in \mathbb{Z}$ with $l_j = 0$ and $u_j = 1$ is called *binary*. In the following four cases we also use other notions for (1):

linear program or LP, if $N = \emptyset$,

integer program or IP, if $C = \emptyset$,

binary mixed integer program, $0-1$ *mixed integer program* or BMIP, if all variables x_j, $j \in N$, are binary,

binary integer program, $0-1$ *integer program* or BIP, if (1) is a BMIP with $C = \emptyset$.

Usually, (1) models a problem arising in some application and the formulation for modeling this problem is not unique. In fact, for the same problem various formulations might exist and the first question is how to select an appropriate formulation. This issue will be discussed in Section 2. Very often however, we do not have our hands on the problem itself but just get the problem formulation as given in (1). In this case, we must extract all relevant information for the solution process from the constraint matrix A, the right-hand side vector b and the objective function c, i.e., we have to perform a structure analysis. This is usually part of the so-called *preprocessing phase* of mixed integer programming solvers and will also be discussed in Section 2. Thereafter, we have a problem, still in the format of (1), but containing more information about the inherit structure of the problem. Secondly, preprocessing also tries to discover and eliminate redundant information from a MIP solver's point of view.

From a complexity point of view mixed integer programming problems belong to the class of $\mathcal{NP}$-hard problems (Garey and Johnson, 1979) which makes it unlikely that efficient, i.e., polynomial time, algorithms for their solution exist. The route one commonly follows to solve an $\mathcal{NP}$-hard problem like (1) to optimality is to attack it from two sides. First, one considers the dual side and determines a lower bound on the objective function by relaxing

the problem. The common basic idea of relaxation methods is to get rid of some part of the problem that causes difficulties. The methods differ in their choice of which part to delete and in the way to reintroduce the deleted part. The most commonly used approach is to relax the integrality constraints to obtain a linear program and reintroduce the integrality by adding cutting planes. This will be the main focus of Section 3. In addition, we will discuss in this section other relaxation methods that delete parts of the constraints and/or variables. Second, we consider the primal side and try to find some good feasible solution in order to determine an upper bound. Unfortunately, very little is done in this respect in general mixed integers solvers, an issue that will be discussed in Section 4.3.

If we are lucky the best lower and upper bounds coincide and we have solved the problem. If not, we have to resort to some enumeration scheme, and the one that is mostly used in this context is the branch-and-bound method. We will discuss branch-and-bound strategies in Section 4 and we will see that they have a big influence on the solution time and quality.

Needless to say that the way described above is not the only way to solve (1), but it is definitely the most used, and often among the most successful. Other approaches include semidefinite programming, combinatorial relaxations, basis reduction, Gomory's group approach, test sets and optimal primal algorithms, see the various articles in this handbook.

2 Formulations and structure analysis

The first step in the solution of an integer program is to find a "right" formulation. Right formulations are of course not unique and they strongly depend on the solution method one wants to use to solve the problem. The method we mainly focus on in this chapter is LP based branch-and-bound. The criterion for evaluating formulations that is mostly used in this context is the tightness of the LP relaxation. If we drop the integrality condition on the variables $x_1, \ldots, x_p$ in problem (1), we obtain the so-called *linear programming relaxation*, or *LP relaxation* for short:

$$\begin{aligned} z_{\mathrm{LP}} = \min \quad & c^T x \\ \text{s.t.} \quad & Ax \left\{ \begin{matrix} \leq \\ = \end{matrix} \right\} b \\ & l \leq x \leq u \\ & x \in \mathbb{R}^n. \end{aligned} \tag{2}$$

For the solution of (2) we have either polynomial (ellipsoid and interior point) or computationally efficient (interior point and simplex) algorithms at hand.

To problem (1) we associate the polyhedron $P_{\mathrm{MIP}} := \mathrm{conv}\{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} : Ax \leq b\}$, i.e., the convex hull of all feasible points for (1). A proof for P_{MIP} being a polyhedron can be found, for instance, in Nemhauser and Wolsey (1988) and Schrijver (1986). In the same way we define the associated polyhedron of problem (2) by $P_{\mathrm{LP}} := \{x \in \mathbb{R}^n : Ax \leq b\}$. Of course, $P_{\mathrm{MIP}} \subseteq P_{\mathrm{LP}}$ and $z_{\mathrm{LP}} \leq z_{\mathrm{MIP}}$, so P_{LP} is a relaxation of P_{MIP}. The crucial requirement in the theory of solving general mixed integer problems is a sufficiently good understanding of the underlying polyhedra in order to tighten this relaxation.

Very often a theoretical analysis is necessary to decide which formulation is superior. There are no general rules such as: "the fewer the number of variables and/or constraints the better the formulation." In the following we discuss as an example a classical combinatorial optimization problem, the Steiner tree problem, which underpins the statement that fewer variables are not always better.

Given an undirected graph $G = (V, E)$ and a node set $T \subseteq V$, a *Steiner tree for T in G* is a subset $S \subseteq E$ of the edges such that $(V(S), S)$ contains a path between s and t for all $s, t \in T$, where $V(S)$ denotes the set of nodes incident to an edge in S. In other words, a Steiner tree is an edge set S that spans T. (Note that by our definition, a Steiner tree might contain circles, in contrast to the usual meaning of the notion *tree* in graph theory.) The *Steiner tree problem* is to find a minimal Steiner tree with respect to some given edge costs $c_e \geq 0$, $e \in E$.

A canonical way to formulate the Steiner tree problem as an integer program is to introduce, for each edge $e \in E$, a variable x_e indicating whether e is in the Steiner tree ($x_e = 1$) or not ($x_e = 0$). Consider the integer program

$$\begin{array}{lll} z_u := \min & c^T x & \\ & x(\delta(W)) \geq 1, & \text{for all } W \subset V,\ W \cap T \neq \emptyset, \\ & & (V \setminus W) \cap T \neq \emptyset, \\ & 0 \leq x_e \leq 1, & \text{for all } e \in E, \\ & x \text{ integer}, & \end{array} \tag{3}$$

where $\delta(X)$ denotes the cut induced by $X \subseteq V$, i.e., the set of edges with one end node in X and one its complement, and $x(F) := \sum_{e \in F} x_e$, for $F \subseteq E$. The first inequalities are called (*undirected*) *Steiner cut inequalities* and the inequalities $0 \leq x_e \leq 1$ *trivial inequalities*. It is easy to see that there is a one-to-one correspondence between Steiner trees in G and 0/1 vectors satisfying the undirected Steiner cut inequalities. Hence, (3) models the Steiner tree problem correctly.

Another way to model the Steiner tree problem is to consider the problem in a directed graph. We replace each edge $\{u, v\} \in E$ by two directed arcs (u, v) and (v, u). Let A denote this set of arcs and $D = (V, A)$ the resulting digraph. We choose some terminal $r \in T$, which will be called the *root*. A *Steiner arborescence* (*rooted at r*) is a set of arcs $S \subseteq A$ such that $(V(S), S)$ contains a directed path from r to t for all $t \in T \setminus \{r\}$. Obviously, there is a one-to-one

correspondence between (undirected) Steiner trees in G and Steiner arborscences in D which contain at most one of two directed arcs (u, v), (v, u). Thus, if we choose arc costs $\vec{c}_{(u,v)} := \vec{c}_{(v,u)} := c_{\{u,v\}}$, for $\{u, v\} \in E$, the Steiner tree problem can be solved by finding a minimal Steiner arborescence with respect to $\vec{c}$. Note that there is always an optimal Steiner arborescence which does not contain an arc and its anti-parallel counterpart, since $\vec{c} \geq 0$. Introducing variables y_a for $a \in A$ with the interpretation $y_a := 1$, if arc a is in the Steiner arborescence, and $y_a := 0$ otherwise, we obtain the integer program

$$\begin{array}{lll} z_d := \min & \vec{c}^T y & \\ & y(\delta^+(W)) \geq 1, & \text{for all } W \subset V, r \in W, \\ & & (V \setminus W) \cap T \neq \emptyset, \\ & 0 \leq y_a \leq 1, & \text{for all } a \in A, \\ & y \text{ integer}, & \end{array} \tag{4}$$

where $\delta^+(X) := \{(u, v) \in A | u \in X, v \in V \setminus X\}$ for $X \subset V$, i.e., the set of arcs with tail in X and head in its complement. The first inequalities are called (*directed*) *Steiner cut inequalities* and $0 \leq y_a \leq 1$ are the *trivial inequalities*. Again, it is easy to see that each 0/1 vector satisfying the directed Steiner cut inequalities corresponds to a Steiner arborescence, and conversely, the incidence vector of each Steiner arborescence satisfies (4). Which of the two models (3) and (4) should be used to solve the Steiner tree problem in graphs?

At first glance, (3) is preferable to (4), since it contains only half the number of variables and the same structure of inequalities. However, it turns out that the optimal value z_d of the LP relaxation of the directed model (4) is greater than or equal to the corresponding value z_u of the undirected formulation (3). Even if the undirected formulation is tightened by the so-called Steiner partition inequalities, this relation holds (Chopra and Rao, 1994). This is astonishing, since the separation problem of the Steiner partition inequalities is difficult ($\mathcal{NP}$-hard), see Grötschel, Monma, and Stoer (1992), whereas the directed Steiner cut inequalities can be separated in polynomial time by max flow computations. Finally, the disadvantage of the directed model that the number of variables is doubled is not really a bottleneck. Since we are minimizing a nonnegative objective function, the variable of one of the two anti-parallel arcs will usually be at its lower bound. If we solve the LP relaxations by the simplex algorithm, it would rarely let this variables enter the basis. Thus, the directed model is much better than the undirected model, though it contains more variables. And in fact, most state-of-the-art solvers for the Steiner tree problem in graphs use formulation (4) or one that is equivalent to (4), see Koch, Martin, and Voß (2001) for further references.

The Steiner tree problem shows that it is not easy to find a tight problem formulation and that often a nontrivial analysis is necessary to come to a good decision.

Once we have decided on some formulation we face the next step, that of eliminating redundant information in (1). This so-called preprocessing step

is very important, in particular, if we have no influence on the formulation step discussed above. In this case it is not only important to eliminate redundant information, but also to perform a structure analysis to extract as much information as possible from the constraint matrix. We will give a nontrivial example concerning block diagonal matrices at the end of this section. Before we come to this point let us briefly sketch the main steps that are usually performed within preprocessing. Most of these options are drawn from Andersen and Andersen (1995), Bixby (1994), Crowder, Johnson, and Padberg (1983), Hoffman and Padberg (1991), Savelsbergh (1994), Suhl and Szymanski (1994). We denote by $s_i \in \{\leq, =\}$ the sense of row l, i.e., (1) reads $\min\{c^T x : Ax \, sb, l \leq x \leq u, x \in \mathbb{Z}^N \times \mathbb{R}^C\}$. We consider the following cases:

Duality fixing. Suppose there is some column j with $c_j \geq 0$ that satisfies $a_{ij} \geq 0$ if $s_i = \text{`}\leq\text{'}$, and $a_{ij} = 0$ if $s_i = \text{`}=\text{'}$ 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 $s_i = \text{`}\leq\text{'}$, $a_{ij} = 0$ if $s_i = \text{`}=\text{'}$ 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.

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

$$\begin{aligned} L_i &= \sum_{j \in P_i} a_{ij} l_j + \sum_{j \in N_i} a_{ij} u_j \\ U_i &= \sum_{j \in P_i} a_{ij} u_j + \sum_{j \in N_i} a_{ij} l_j \end{aligned} \tag{5}$$

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) $s_i = \text{`}=\text{'}$ and $L_i > b_i$ or $U_i < b_i$
   (b) $s_i = \text{`}\leq\text{'}$ and $L_i > b_i$

   In these cases the problem is infeasible.

2. Forcing row:
   (a) $s_i = \text{`}=\text{'}$ and $L_i = b_i$ or $U_i = b_i$
   (b) $s_i = \text{`}\leq\text{'}$ and $L_i = b_i$

   Here, all variables in P_i can be fixed to their lower (upper) bound and all variables in N_i to their upper (lower) bound when $L_i = b_i (U_i = b_i)$. Row i can be deleted afterwards.

3. Redundant row:
   (a) $s_i = \text{`}\leq\text{'}$ 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

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

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

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 commonly be in the basis and are thus useful in finding a starting basis. For mixed integer programs however, it is better in general 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`[1]):

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

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.

Coefficients reduction. The row bounds in (5) can also be used to reduce the absolute value of coefficients of binary variables. Consider some row i with $s_i = \text{`}\leq\text{'}$ and let x_j be a binary variable with $a_{ij} \neq 0$.

$$\text{If} \begin{cases} a_{ij} < 0, U_i + a_{ij} < b_i, & \text{set} \quad \bar{a}'_{ij} = b_i - U_i, \\ a_{ij} > 0, U_i - a_{ij} < b_i, & \text{set} \quad \begin{cases} \bar{a}'_{ij} = U_i - b_i, \\ \bar{b}_i = U_i - a_{ij}, \end{cases} \end{cases} \tag{6}$$

[1] `Miplib` is a public available test set of real-world mixed integer programming problems (Bixby, Ceria, McZeal, and Savelsbergh, 1998).

where $\bar{a}'_{ij}$ denotes the new reduced coefficient. Consider the following inequality from example p0033 in the `Miplib`:

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

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

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

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

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

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

$$\frac{b_i - a_{ij}x_j}{a_{ik}}. \tag{7}$$

In case x_k is binary or integer, the substitution is only possible, if the term (7) 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 variables 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 (i.e., nonzero coefficients) in the matrix A that increases computer memory demand and lowers the computational speed of the simplex algorithm. Hence, aggregation is usually restricted to constraints and columns with small support.

Disaggregation. Disaggregation of columns, to our knowledge, is not an issue in preprocessing of mixed integer programs, 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 (Gondzio, 1997).
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)

$$x_{85} + x_{90} + x_{95} + x_{100} + x_{217} + x_{222} + x_{227} + x_{232} - 8x_{246} \leq 0 \quad (8)$$

where all variables involved are binary. The inequality says that whenever one of the variables x_i with $i \in S := \{85, 90, 95, 100, 217, 222, 227, 232\}$ is one, x_{246} must also be one. This fact can also be expressed by replacing (8) by the following eight inequalities:

$$x_i - x_{246} \leq 0 \quad \text{for all } i \in S. \quad (9)$$

This formulation is tighter in the following sense: 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, see the next section, this problem is not really an issue, because the inequalities in (9) can be generated on demand.

Probing. Probing is sometimes used in general mixed integer programming codes, see, for instance, Savelsbergh (1994), Suhl and Szymanski (1994). The idea is to set some binary variable temporarily to zero or one and try to deduce further or stronger inequalities from that. These implications can be expressed in inequalities as follows:

$$\begin{aligned} (x_j = 1 \Rightarrow x_i = \alpha) &\Rightarrow \begin{cases} x_i \geq l_i + (\alpha - l_i)x_j \\ x_i \leq u_i - (u_i - \alpha)x_j \end{cases} \\ (x_j = 0 \Rightarrow x_i = \alpha) &\Rightarrow \begin{cases} x_i \geq \alpha - (\alpha - l_i)x_j \\ x_i \leq \alpha + (u_i - \alpha)x_j. \end{cases} \end{aligned} \quad (10)$$

As an example, suppose we set variable x_{246} temporarily to zero in (8). This implies that $x_i = 0$ for all $i \in S$. Applying (10) we deduce the inequality

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

for all $i \in S$, which is exactly (9). For further aspects of probing refer to Atamtürk, Nemhauser, and Savelsbergh (2000), where probing is used

for the construction of conflict graphs to strengthen the LP relaxation, Johnson, Nemhauser, and Savelsbergh (2000), where probing is applied to improve the coefficients of the given inequalities, and Savelsbergh (1994), where a comprehensive study of probing is provided.

Besides the cases described, there are trivial ones like empty rows, empty, infeasible, and fixed columns, parallel rows and singleton rows or columns that we refrain from discussing here. One hardly believes at this point that such examples or some of the above cases really appear in mixed integer programming formulations, because better formulations are straight-forward to derive. But such formulations do indeed come up and mixed integer programming solvers must be able to handle them. Reasons for their existence are that formulations are often made by nonexperts or are sometimes generated automatically by some matrix generating program.

In general, all these tests are iteratively applied until all of them fail. 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 Section 4. 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. Martin (1998), for instance, performs some computational tests for the instances in the `Miplib`. His results show that preprocessing reduces the problem sizes in terms of number of rows, columns, and nonzeros by around 10% on average. The time spent in preprocessing is negligible (below one per mill). It is interesting to note that for some problems presolve is indispensable for their solution. For example, problem fixnet6 in the `Miplib` is an instance, on which most solvers fail without preprocessing, but with presolve the instance turns out to be very easy. Further results on this subject can be found in Savelsbergh (1994).

Observe also that the preprocessing steps discussed so far consider just one single row or column at a time. The question comes up, whether one could gain something by looking at the structure of the matrix as a whole. This is a topic of computational linear algebra where one tries on one side to speed-up algorithms for matrices in special forms and on the other hand tries to develop algorithms that detect certain forms after reordering columns and/or rows. It is interesting to note that the main application area in this field are matrices arising from PDE systems. Very little has been done in connection with mixed integer programs. In the following we discuss one case, which shows that there might be more potential for MIPs.

Consider a matrix in a so-called bordered block diagonal form as depicted in Fig. 1. Suppose the constraint matrix of (1) has such a form and suppose in addition that there are just a few or even no coupling constraints. In the latter case the problem decomposes into a number of blocks many independent problems, which can be solved much faster than the original problem. Even if

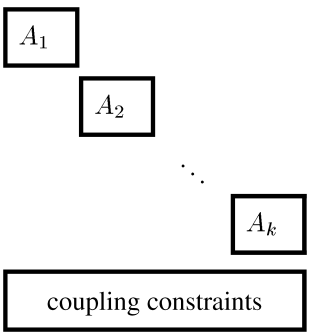

Fig. 1. Matrix in bordered block diagonal form.

there are coupling constraints this structure might help for instance to derive new cutting planes. The question arises whether MIPs have such a structure, possibly after reordering columns and rows? There are some obvious cases, where the matrix is already in this form (or can be brought into it), such as multi-commodity flow problems, multiple knapsack problems or other packing problems. But, there are problems where a bordered block diagonal form is hidden in the problem formulation (1) and can only be detected after reordering columns and rows. Borndörfer, Ferreira, and Martin (1998) have analyzed this question and checked whether matrices from MIPs can be brought into this form. They have tested various instances, especially problems whose original formulation is not in bordered block diagonal form, and it turns out that many problems have indeed such a form. Even more, the heuristics developed for detecting such a form are fast enough to be incorporated into preprocessing of a MIP solver. Martin and Weismantel (Martin, 1998; Martin and Weismantel, 1998) have developed cutting planes that exploit bordered block diagonal form and the computational results for this class of cutting planes are very promising. Of course, this is just a first step of exploiting special structures of MIP matrices and more needs to be done in this direction.

3 Relaxations

In obtaining good or optimal solutions of (1) one can approach it in two different ways: from the *primal* side by computing feasible solutions (mostly by heuristics) or from the dual side by determining good lower bounds. This is done by relaxing the problem. We consider three different types of relaxation ideas. The first and most common is to relax the integrality constraints and to find cutting planes that strengthen the resulting LP relaxation. This is the topic of Section 3.1. In Section 3.2 we sketch further well-known approaches, Lagrangean relaxation as well as Dantzig–Wolfe and Benders' decomposition.

The idea of these approaches is to delete part of the constraint matrix and reintroduce it into the problem either in the objective function or via column generation or cutting planes, respectively.

3.1 Cutting planes

The focus of this section is on describing cutting planes that are used in general mixed integer programming solvers. Mainly, we can classify cutting planes generating algorithms in two groups: one is exploiting the structure of the underlying mixed integer program, the other not. We first take a closer look on the latter group, in which we find the so-called Gomory cuts, mixed integer rounding cuts and lift-and-project cuts.

Suppose we want to solve the mixed integer program (1), where we assume for simplicity that we have no equality constraints and that $N = \{1, \ldots, p\}$ and $C = \{p+1, \ldots, n\}$. Note that if $x^* = (x_1^*, \ldots, x_n^*)$ is an optimal solution of (2) and x^* is in $\mathbb{Z}^p \times \mathbb{R}^{n-p}$, then it is already an optimal solution of (1) and we are done. But this is unlikely to happen after just solving the relaxation. It is more realistic to expect that some (or even all) of the variables $x_1^*, \ldots, x_p^*$ are not integral. In this case there exists at least one inequality $a^T x \le \beta$ that is feasible for P_{MIP} but not satisfied by x^*. From a geometric point of view, x^* is cut off by the hyperplane $a^T x \le \beta$ and therefore $a^T x \le \beta$ is called a *cutting plane*. The problem of determining whether x^* is in P_{MIP} and if not of finding such a cutting plane is called the *separation problem*. If we find a cutting plane $a^T x \le \beta$, we add it to the problem (2) and obtain

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \le b \\ & a^T x \le \beta \\ & x \in \mathbb{R}^n, \end{array} \tag{11}$$

which strengthens (2) in the sense that $P_{\mathrm{LP}} \subset P_{\mathrm{LP}^1} \subseteq P_{\mathrm{MIP}}$, where $P_{\mathrm{LP}^1} := \{x : Ax \le b, a^T x \le \beta\}$ is the associated polyhedron of (11). Note that the first inclusion is strict by construction.

The process of solving (11) and finding a cutting plane is now iterated until the solution is in $\mathbb{Z}^p \times \mathbb{R}^{n-p}$ (this will be the optimal solution of (1)). Let us summarize the cutting plane algorithm discussed so far:

Algorithm 1. (*Cutting plane*)

1. *Let* $k := 0$ *and* LP^0 *the linear programming relaxation of the mixed integer program* (1).
2. *Solve* LP^k. *Let* $\tilde{x}^k$ *be an optimal solution.*
3. *If* $\tilde{x}^k$ *is in* $\mathbb{Z}^p \times \mathbb{R}^{n-p}$, *stop;* $\tilde{x}^k$ *is an optimal solution of* (1).

4. *Otherwise, find a linear inequality, that is satisfied by all feasible mixed integer points of* (1), *but not by* $\tilde{x}^k$.
5. *Add this inequality to* LP^k *to obtain* LP^{k+1}.
6. *Increase* k *by one and go to Step* 2.

The remaining of this section is devoted to the question on how to find good cutting planes.

3.1.1 Gomory integer cuts

We start with the pure integer case, i.e., $p = n$ in problem (1). The cutting plane algorithm we present in the sequel is based on simple integer rounding and makes use of information given by the simplex algorithm. Hereto we transform the problem into the standard form by adding slack variables and substituting unbounded variables $x_i := x_i^+ - x_i^-$ by two variables $x_i^+, x_i^- \geq 0$ that are bounded from below. Summing up, we turned (1) into a problem with the following structure:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \in \mathbb{Z}_+^n, \end{array} \tag{12}$$

with $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$. (Note that this A, c and x may differ from those in (1).) We denote the associated polyhedron by $P_{\mathrm{IP}}^{\mathrm{St}} := \mathrm{conv}\{x \in \mathbb{Z}_+^n : Ax = b\}$. Let x^* be an optimal solution of the LP relaxation of (12). We partition x^* into two subvectors x_B^* and x_N^*, where $B \subset \{1, \ldots, n\}$ is a basis of A, i.e., A_B nonsingular (regular), with

$$x_B^* = A_B^{-1} b - A_B^{-1} A_N x_N^* \geq 0 \tag{13}$$

and $x_N^* = 0$ for the nonbasic variables where $N = \{1, \ldots, n\} \backslash B$. (Note that this N completely differs from the N used in (1).) If x^* is integral, we have found an optimal solution of (12). Otherwise, at least one of the values in x_B^* must be fractional. So we choose $i \in B$ such that $x_i^* \notin \mathbb{Z}$. From (13) we get the following expression for the i-th variable of x_B:

$$A_i.^{-1} b = \sum_{j \in N} A_i.^{-1} A_{.j} x_j + x_i, \tag{14}$$

where $A_i.^{-1}$ denotes the i-th row of A^{-1} and $A_{.j}$ the j-th column of A, respectively. We set $\bar{b}_i := A_i.^{-1} b$ and $\bar{a}_{ij} := A_i.^{-1} A_{.j}$ for short. Since $x_j \geq 0$ for all j,

$$x_i + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j \leq x_i + \sum_{j \in N} \bar{a}_{ij} x_j = \bar{b}_i. \tag{15}$$

We can round down the right-hand side, since x is assumed to be integral and nonnegative, and thus the left-hand side in (15) is integral. So we obtain

$$x_i + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor \bar{b}_i \rfloor. \tag{16}$$

This inequality is valid for all integral points of $P_{\mathrm{IP}}^{\mathrm{St}}$, but it cuts off x^*, since $x_i^* = \bar{b}_i \notin \mathbb{Z}$, $x_j^* = 0$ for all $j \in N$ and $\lfloor \bar{b}_i \rfloor < \bar{b}_i$. Furthermore, all values of (16) are integral. After introducing another slack variable we add it to (12) still fulfilling the requirement that all values in the constraint matrix, the right-hand side and the new slack variable have to be integral. Named after their inventor, inequalities of this type are called *Gomory cuts* (Gomory, 1958, 1960). Gomory showed that an integer optimal solution is found after repeating these steps a finite number of times.

3.1.2 Gomory mixed integer cuts

The previous approach of generating valid inequalities fails if both integer and continuous variables are present. It fails, because rounding down the right-hand side may cut off some feasible points of $P_{\mathrm{MIP}}^{\mathrm{St}} := \mathrm{conv}\{x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p} : Ax = b\}$, if x cannot be assumed to be integral. For the general mixed-integer case, we describe three different methods to obtain valid inequalities. They are all more or less based on the following disjunctive argument.

Lemma 3.2. *Let P and Q be two polyhedra in $\mathbb{R}_+^n$ and $a^T x \leq \alpha$, $b^T x \leq \beta$ valid inequalities for P and Q respectively. Then*

$$\sum_{i=1}^{n} \min(a_i, b_i) x_i \leq \max(\alpha, \beta)$$

is valid for conv $(P \cup Q)$.

We start again with a mixed integer problem in standard form, but this time with $p < n$, i.e.,

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}. \end{array} \tag{17}$$

Let $P_{\mathrm{MIP}}^{\mathrm{St}}$ be the convex hull of all feasible solutions of (17). Consider again (14), where B is a basis, x_i, $i \in B$, is an integer variable and $\bar{b}_i, \bar{a}_{ij}$ are defined accordingly. We divide the set N of nonbasic variables in $N^+ := \{j \in N : \bar{a}_{ij} \geq 0\}$ and $N^- := N \backslash N^+$. As we already mentioned, every feasible x of (17) satisfies $x_B = A_B^{-1} b - A_B^{-1} A_N x_N$, hence

$$\bar{b}_i - \sum_{j \in N} \bar{a}_{ij} x_j \in \mathbb{Z}$$

and there exists $k \in \mathbb{Z}$ such that

$$\sum_{j \in N} \bar{a}_{ij} x_j = f(\bar{b}_i) + k, \tag{18}$$

where $f(\alpha) := \alpha - \lfloor \alpha \rfloor$ for $\alpha \in \mathbb{R}$. In order to apply the disjunctive argument, we distinguish the following two cases, $\sum_{j \in N} \bar{a}_{ij} x_j \geq 0$ and $\sum_{j \in N} \bar{a}_{ij} x_j \leq 0$. In the first case

$$\sum_{j \in N^+} \bar{a}_{ij} x_j \geq f(\bar{b}_i)$$

follows. In the second case we get

$$\sum_{j \in N^-} \bar{a}_{ij} x_j \leq f(\bar{b}_i) - 1 \leq 0$$

or, equivalently,

$$-\frac{f(\bar{b}_i)}{1 - f(\bar{b}_i)} \sum_{j \in N^-} \bar{a}_{ij} x_j \geq f(\bar{b}_i).$$

Now we apply the disjunctive argument to the disjunction $P := P^{\mathrm{St}}_{\mathrm{MIP}} \cap \{x : \sum_{j \in N} \bar{a}_{ij} x_j \geq 0\}$ and $Q := P^{\mathrm{St}}_{\mathrm{MIP}} \cap \{x : \sum_{j \in N} \bar{a}_{ij} x_j \leq 0\}$. Because of $\max(\bar{a}_{ij}, 0) = \bar{a}_{ij}$ for $j \in N^+$ and $\max(-f(\bar{b}_i)/(1 - f(\bar{b}_i))\bar{a}_{ij}, 0) = -f(\bar{b}_i)/(1 - f(\bar{b}_i))\bar{a}_{ij}$ for $j \in N^-$ we obtain by applying Lemma 3.2 the valid inequality for $P^{\mathrm{St}}_{\mathrm{MIP}}$

$$\sum_{j \in N^+} \bar{a}_{ij} x_j - \frac{f(\bar{b}_i)}{1 - f(\bar{b}_i)} \sum_{j \in N^-} \bar{a}_{ij} x_j \geq f(\bar{b}_i), \tag{19}$$

which cuts off x^*, since all nonbasic variables are zero. It is possible to strengthen inequality (19) in the following way. Observe that the derivation does not change, if we add integer multiples to those variables x_j, $j \in N$, that are integral (only the value of k in (18) might change). By doing this we may put the coefficient of each integer variable x_j either in the set N^+ or N^-. If we put it in N^+, the derivation of the inequality yields $\bar{a}_{ij}$ as coefficient for x_j. Thus the best possible coefficient after adding integer multiples is $f(\bar{a}_{ij})$, the difference between the right-hand and left-hand side in (19) is now as small as possible. In N^- the final coefficient is $-f(\bar{b}_i)/(1 - f(\bar{b}_i))\bar{a}_{ij}$, so the smallest difference is achieved by the factor $f(\bar{b}_i)(1 - f(\bar{a}_{ij}))/(1 - f(\bar{b}_i))$. We still have the freedom to select between N^+ and N^-. We obtain the best possible coefficients by using $\min(f(\bar{a}_{ij}), f(\bar{b}_i)(1 - f(\bar{a}_{ij}))/(1 - f(\bar{b}_i)))$. Putting

all this together yields Gomory's mixed integer cut (Gomory, 1960):

$$\sum_{\substack{j\in N,\, j\le p\\ f(\bar{a}_{ij})\le f(\bar{b}_i)}} f(\bar{a}_{ij})x_j + \sum_{\substack{j\in N,\, j\le p\\ f(\bar{a}_{ij})> f(\bar{b}_i)}} \frac{f(\bar{b}_i)(1-f(\bar{a}_{ij}))}{1-f(\bar{b}_i)}x_j + \sum_{j\in N^+,\, j>p} \bar{a}_j x_j - \sum_{j\in N^-,\, j>p} \frac{f(\bar{b}_i)}{1-f(\bar{b}_i)}\bar{a}_j x_j \ge f(\bar{b}_i). \tag{20}$$

Gomory (1960) showed that an algorithm based on iteratively generated inequalities of this type solves (1) after a finite number of steps, if the objective function value $c^T x$ is integer for all $x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}$ with $Ax = b$.

In the derivation of Gomory's mixed integer cuts we followed the original path of Gomory (1960). Having mixed-integer-rounding cuts at hand, we can give another proof for their validity in just one single line at the end of the next section.

Though Gomory's mixed integer cuts have been known since the sixties, their computational breakthrough came in the nineties with the paper by Balas, Ceria, Cornuéjols, and Natraj (1996). In the meantime they are incorporated in many MIP solvers, see, for instance Bixby, Fenelon, Guand, Rothberg, and Wunderling (1999). Note that Gomory's mixed integer cuts can always be applied, as the separation problem for the optimal LP solution is easy. However, adding these inequalities might cause numerical difficulties, see the discussion in Padberg (2001).

3.1.3 Mixed-integer-rounding cuts

We start developing the idea of this kind of cutting planes by considering the subset $X := \{(x, y) \in \mathbb{Z} \times \mathbb{R}_+ : x - y \le b\}$ of $\mathbb{R}^2$ with $b \in \mathbb{R}$. We define two disjoint subsets $P := \text{conv}(X \cap \{(x, y) : x \le \lfloor b \rfloor\})$ and $Q := \text{conv}(X \cap \{(x, y) : x \ge \lfloor b \rfloor + 1\})$ of $\text{conv}(X)$. For P the inequalities $x - \lfloor b \rfloor \le 0$ and $0 \le y$ are valid and therefore every linear combination of them is also valid. Hence, if we multiply them by $1 - f(b)$ and 1 respectively, we obtain

$$(x - \lfloor b \rfloor)(1 - f(b)) \le y.$$

For Q we scale the valid inequalities $-(x - \lfloor b \rfloor) \le -1$ and $x - y \le b$ with weights $f(b)$ and 1 to get

$$(x - \lfloor b \rfloor)(1 - f(b)) \le y.$$

Now the disjunctive argument, Lemma 3.2, implies that $(x - \lfloor b \rfloor)(1 - f(b)) \le y$, or equivalently:

$$x - \frac{1}{1 - f(b)} y \le \lfloor b \rfloor \tag{21}$$

is valid for $\text{conv}(P \cup Q) = \text{conv}(X)$.

From this basic situation we change now to more general settings. Consider the mixed integer set $X := \{(x, y) \in \mathbb{Z}_+^p \times \mathbb{R}_+ : a^T x - y \leq b\}$ with $a \in \mathbb{R}^p$ and $b \in \mathbb{R}$. We define a partition of $\{1, \ldots, n\}$ by $N^1 := \{i \in \{1, \ldots, n\} : f(a_i) \leq f(b)\}$ and $N^2 := \{1, \ldots, n\} \backslash N^1$. With this setting we obtain

$$\sum_{i \in N^1} \lfloor a_i \rfloor x_i + \sum_{i \in N^2} a_i x_i - y \leq a^T x - y \leq b.$$

Now let $w := \sum_{i \in N^1} \lfloor a_i \rfloor x_i + \sum_{i \in N^2} \lceil a_i \rceil x_i \in \mathbb{Z}$ and $z := y + \sum_{i \in N^2} (1 - f(a_i))$ $x_i \geq 0$, then we obtain (remark that $\lceil a_i \rceil - \lfloor a_i \rfloor \leq 1$)

$$\begin{aligned} w - z &= \sum_{i \in N^1} \lfloor a_i \rfloor x_i + \sum_{i \in N^2} \lceil a_i \rceil x_i - \sum_{i \in N^2} (1 - a_i + \lfloor a_i \rfloor) x_i - y \\ &\leq \sum_{i \in N^1} \lfloor a_i \rfloor x_i + \sum_{i \in N^2} a_i x_i - y \leq b. \end{aligned}$$

and (21) yields

$$w - \frac{1}{1 - f(b)} z \leq \lfloor b \rfloor.$$

Substituting w and z gives

$$\sum_{i \in N^1} \lfloor a_i \rfloor x_i + \sum_{i \in N^2} \left(\lceil a_i \rceil - \frac{1 - f(a_i)}{1 - f(b)} \right) x_i - \frac{1}{1 - f(b)} y \leq \lfloor b \rfloor.$$

Easy computation shows that this is equivalent to

$$\sum_{i=1}^{p} \left(\lfloor a_i \rfloor + \frac{\max(0, f(a_i) - f(b))}{1 - f(b)} \right) x_i - \frac{1}{1 - f(b)} y \leq \lfloor b \rfloor.$$

Thus we have shown that this is a valid inequality for $\text{conv}(X)$, the *mixed integer rounding* (*MIR*) *inequality*.

From MIR inequalities one can easily derive Gomory's mixed integer cuts. Consider the set $X := \{(x, y^-, y^+) \in \mathbb{Z}_+^p \times \mathbb{R}_+^2 | a^T x + y^+ - y^- = b\}$, then $a^T x - y^- \leq b$ is valid for X and the computations shown above now yield

$$\sum_{i=1}^{p} \left(\lfloor a_i \rfloor + \frac{\max(0, f(a_i) - f(b))}{1 - f(b)} \right) x_i - \frac{1}{1 - f(b)} y^- \leq \lfloor b \rfloor$$

as a valid inequality. Subtracting $a^T x + y^+ - y^- = b$ gives Gomory's mixed integer cut.

Nemhauser and Wolsey (1990) discuss MIR inequalities in a more general setting. They prove that MIR inequalities provide a complete description for any mixed 0–1 polyhedron. Marchand and Wolsey (Marchand, 1998; Marchand and Wolsey, 2001) show the computational merits of MIR inequalities in solving general mixed integer programs.

3.1.4 Lift-and-project cuts

The cuts presented here only apply to 0–1 mixed integer problems. The idea of "lift and project" is to find new inequalities not in the original space but in a higher dimensional (lifting). By projecting these inequalities back to the original space tighter inequalities can be obtained. In literature many different ways to lift and to project back can be found (Balas, Ceria, and Cornuéjols, 1993; Bienstock and Zuckerberg, 2003; Lasserre, 2001; Lovász and Schrijver, 1991; Sherali and Adams, 1990). The method we want to review in detail is due to Balas et al. (1993, 1996). It is based on the following observation:

Lemma 3.3. *If* $\alpha + a^T x \geq 0$ *and* $\beta + b^T x \geq 0$ *are valid for a polyhedron* P, *then* $(\alpha + a^T x)(\beta + b^T x) \geq 0$ *is also valid for* P.

We consider a 0–1 program in the form of (1) having w.l.o.g. no equality constraints, in which the system $Ax \leq b$ already contains the trivial inequalities $0 \leq x_i \leq 1$ for all $i \in \{1, \ldots, p\}$. The following steps give an outline of the lift-and-project procedure:

Algorithm 4. (*Lift-and-project*)

1. *Choose an index* $j \in \{1, \ldots, p\}$.
2. *Multiply each inequality of* $Ax \leq b$ *once by* x_j *and once by* $1 - x_j$ *giving the new* (*nonlinear*) *system*:

$$\begin{aligned} (Ax)x_j &\leq bx_j \\ (Ax)(1 - x_j) &\leq b(1 - x_j) \end{aligned} \tag{22}$$

3. *Lifting: replace* $x_i x_j$ *by* y_i *for* $i \in \{1, \ldots, n\} \setminus \{j\}$ *and* x_j^2 *by* x_j. *The resulting system of inequalities is again linear and finite and the set of its feasible points* $L_j(P)$ *is therefore a polyhedron.*
4. *Projection: project* $L_j(P)$ *back to the original space by eliminating all variables* y_i. *Call the resulting polyhedron* P_j.

In Balas et al. (1993) it is proven that $P_j = \mathrm{conv}(P \cap \{x \in \mathbb{R}^n : x_j \in \{0, 1\})$, i.e., the j-th component of each vertex of P_j is either zero or one. Moreover, it is shown that a repeated application of Algorithm 4 on the first p variables yields

$$((P_1)_2 \ldots)_p = \mathrm{conv}(P \cap \{x \in \mathbb{R}^n : x_1, \ldots, x_p \in \{0, 1\}\}) = P_{\mathrm{MIP}}.$$

In fact, this result does not depend on the order in which one applies lift-and-project. Every permutation of $\{1, \ldots, p\}$ yields P_{MIP}.

The crucial step we did not describe up to now is how to carry out the projection (Step 4). As $L_j(P)$ is a polyhedron, there exists matrices D, B and a vector d such that $L_j(P) = \{(x,\ y) : Dx + By \leq d\}$. Thus we can describe the (orthogonal-) projection of $L_j(P)$ onto the x-space by

$$P_j = \{x \in \mathbb{R}^n : (u^T D)x \leq u^T d \quad \text{for all } u \geq 0, u^T B = 0\}.$$

Now that we are back in our original problem space, we can start finding valid inequalities by solving the following linear program for a given fractional solution x^* of the underlying mixed integer problem:

$$\begin{array}{ll} \max & u^T(Dx^* - d) \\ \text{s.t.} & u^T B = 0 \\ & u \in \mathbb{R}^n_+. \end{array} \tag{23}$$

The set $C := \{u \in \mathbb{R}^n_+ : u^T B = 0\}$ in which we are looking for the optimum is a pointed polyhedral cone. The optimum is either 0, if the variable x_j is already integral, or the linear program is unbounded (infinity). In the latter case let $u^* \in C$ be an extreme ray of the cone in which direction the linear program (23) is unbounded. Then u^* will give us the cutting plane $(u^*)^T Dx \leq (u^*)^T d$ that indeed cuts off x^*.

Computational experiences with lift-and-project cuts to solve real-world problems are discussed in Balas et al. (1993, 1996).

3.1.5 Knapsack inequalities

The cutting planes discussed so far have one thing common: they do not make use of the special structures of the given problem. In this section we want to generate valid inequalities by investigating the underlying combinatorial problem. The inequalities that are generated in this way are usually stronger in the sense that one can prove that they induce high-dimensional faces, often facets, of the underlying polyhedron.

We start again with the pure integer case. A knapsack problem is a 0–1 integer problem with just one inequality $a^T x \leq \beta$. Its polytope, the 0–1 knapsack polytope, is the following set of points:

$$P_K(N, a, \beta) := \text{conv}\left\{x \in \{0, 1\}^N : \sum_{j \in N} a_j x_j \leq \beta\right\}$$

with a finite set N, weights $a \in \mathbb{Z}^N_+$ and some capacity $\beta \in \mathbb{Z}_+$.

Observe that each inequality of a 0–1 program gives rise to a 0–1 knapsack polytope. And thus each valid inequality known for the knapsack polytope can be used to strengthen the 0–1 program. In the sequel we derive some known inequalities for the 0–1 knapsack polytope that are also useful for solving general 0–1 integer problems.

Cover inequalities. A subset $C \subseteq N$ is called a *cover* if $\sum_{j \in C} a_j > \beta$, i.e., the sum of the weights of all items in C is bigger than the capacity of the knapsack. To each cover, we associate the cover inequality

$$\sum_{j \in C} x_j \leq |C| - 1,$$

a valid inequality for $P_K(N, a, \beta)$. If the underlying cover C is *minimal*, i.e., $C \subset N$ is a cover and for every $s \in C$ we have $\sum_{j \in S \setminus \{s\}} a_j \leq \beta$, the inequality defines a facet of $P_K(C, a, \beta)$, i.e., the dimension of the face that is induced by the inequality is one less than the dimension of the polytope. Nonminimal cover only give faces, but not facets. Indeed, if a cover is not minimal, the corresponding cover inequality is superfluous, because it can be expressed as a sum of minimal cover inequalities and some upper bound constraints. Minimal cover inequalities might be strengthened by a technique called lifting that we present in detail in the next section.

$(1, k)$-Configuration inequalities. Padberg (1980) introduced this class of inequalities. Let $S \subset N$ be a set of items that fits into the knapsack, $\sum_{j \in S} a_j \leq \beta$, and suppose there is another item $z \in N \setminus S$ such that $\tilde{S} \cup \{z\}$ is a minimal cover for every $\tilde{S} \subset S$ with cardinality $|\tilde{S}| = k$. Then (S, z) is called a $(1, k)$-*configuration*. We derive the following inequality

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

which we call $(1, k)$-*configuration* inequality. They are connected to minimal cover inequalities in the following way: a minimal cover S is a $(1, |S| - 1)$-configuration and a $(1, k)$-configuration with respect to $(S, \{z\})$ with $k = |S|$ is a minimal cover. Moreover, one can show that $(1, k)$-configuration inequalities define facets of $P_K(S \cup \{z\}, a, \beta)$.

Extended weight inequalities. Weismantel (1997) generalized minimal cover and $(1, k)$-configuration inequalities. He introduced extended weight inequalities which include both classes of inequalities as special cases. Denote $a(T) := \sum_{j \in T} a_j$ and consider a subset $T \subset N$ such that $a(T) < \beta$. With $r := \beta - a(T)$, the inequality

$$\sum_{i \in T} a_i x_i + \sum_{i \in N \setminus T} \max(a_i - r, 0) x_i \leq a(T) \tag{24}$$

is valid for $P_K(N, a, \beta)$. It is called a *weight inequality with respect to* T. The name *weight inequality* reflects the fact that the coefficients of the items in T equal their original weights and the number $r := \beta - a(T)$ corresponds to the remaining capacity of the knapsack when $x_j = 1$ for all $j \in T$. There is a natural way to extend weight inequalities by (i) replacing

the original weights of the items by relative weights and (ii) using the method of sequential lifting that we outline in Section 3.1.8.
Let us consider a simple case by associating a weight of one to each of the items in T. Denote by S the subset of $N\backslash T$ such that $a_j \geq r$ for all $j \in S$. For a chosen permutation $\pi_1, \ldots, \pi_{|S|}$ of S we apply sequential lifting, see Section 3.1.8, and obtain lifting coefficients w_j, $j \in S$ such that

$$\sum_{j \in T} x_j + \sum_{j \in S} w_j x_j \leq |T|,$$

is a valid inequality for $P_K(N, a, \beta)$, called the (*uniform*) *extended weight inequality*. They already generalize minimal cover and $(1, k)$-configuration inequalities and can be generalized themselves to inequalities with arbitrary weights in the starting set T, see Weismantel (1997).

The separation of minimal cover inequalities is widely discussed in the literature. The complexity of cover separation has been investigated in Ferreira (1994), Gu, Nemhauser, and Savelsbergh (1998), Klabjan, Nemhauser, Tovey (1998), whereas algorithmic and implementation issues are treated among others in Crowder, Johnson, and Padberg (1983), Gu, Nemhauser, and Savelsbergh (1998), Hoffman and Padberg (1991), Van Roy and Wolsey (1987), Zemel (1989). The ideas and concepts suggested to separate cover inequalities basically carry over to extended weight inequalities. Typical features of a separation algorithm for cover inequalities are: fix all variables that are integers, find a cover (in the extended weight case some subset T) usually by some greedy-type heuristics, and lift the remaining variables sequentially.

Cutting planes derived from knapsack relaxations can sometimes be strengthened if special ordered set (SOS) inequalities $\sum_{j \in Q} x_j \leq 1$ for some $Q \subseteq N$ are available. In connection with a knapsack inequality these constraints are also called *generalized upper bound constraints* (*GUBs*). It is clear that by taking the additional SOS constraints into account stronger cutting planes may be derived. This possibility has been studied in Crowder, Johnson, and Padberg (1983), Gu, Nemhauser, and Savelsbergh (1998), Johnson and Padberg (1981), Nemhauser and Vance (1994), Wolsey (1990).

From pure integer knapsack problems we switch now to mixed 0–1 knapsacks, where some continuous variables appear. As we will see, the concept of covers is also useful in this case to describe the polyhedral structure of the associated polytopes. Consider the mixed 0–1 knapsack set

$$P_S(N, a, \beta) = \left\{ (x, s) \in \{0, 1\}^N \times \mathbb{R}_+ : \sum_{j \in N} a_j x_j - s \leq \beta \right\}$$

with nonnegative coefficients, i.e., $a_j \geq 0$ for $j \in N$ and $\beta \geq 0$.

Now let $C \subset N$ be a cover and $\lambda := \sum_{j \in C} a_j - \beta > 0$. Marchand and Wolsey (1999) showed that the inequality

$$\sum_{j \in C} \min(a_j, \lambda) x_j - s \leq \sum_{j \in C} \min(a_j, \lambda) - \lambda \tag{25}$$

is valid for $P_S(N, a, \beta)$. Moreover, this inequality defines a facet of $P_S(C, a, \beta)$. This result marks a contrast to the pure 0–1 knapsack case, where only minimal covers induce facets. Computational aspects of these inequalities are discussed in Marchand (1998), Marchand and Wolsey (1999).

Cover inequalities also appear in other contexts. In Ceria, Cordier, Marchand, and Wolsey (1998) cover inequalities are derived for the knapsack set with general integer variables. Unfortunately, in this case, the resulting inequalities do not define facets of the convex hull of the knapsack set restricted to the variables defining the cover. More recently, the notion of cover has been used to define families of valid inequalities for the complementarity knapsack set (de Farias, Johnson, and Nemhauser, 2002). By lifting continous variables new inequalities are developed in Richard, de Farias, and Nemhauser (2001) that extended (25). Atamtürk (2001) studies the convex hull of feasible solutions for a single constraint taken from a mixed-integer programming problem. No sign restrictions are imposed on the coefficients and the variables are not necessarily bounded, thus mixed 0–1 knapsacks are contained as a special case. It is still possible to obtain strong valid inequalities that may be useful for general mixed-integer programming.

3.1.6 Flow cover inequalities

From (mixed) knapsack problems with only one inequality we now turn to more complex polyhedral structures. Consider within a capacitated network flow problem, some node with a set of ingoing arcs N. Each inflow arc $j \in N$ has a capacity a_j. By y_j we denote the (positive) flow that is actually on arc $j \in N$. Moreover, the total inflow (i.e., sum of all flows on the arcs in N) is bounded by $b \in \mathbb{R}_+$. Then the (flow) set of all feasible points of this problem is given by

$$X = \left\{ (x, y) \in \{0, 1\}^N \times \mathbb{R}_+^N : \sum_{j \in N} y_j \leq b, y_j \leq a_j x_j, \ \forall j \in N \right\}. \tag{26}$$

We want to demonstrate how to use the mixed knapsack inequality (25) to derive new inequalities for the polyhedron $\mathrm{conv}(X)$. Let $C \subset N$ be a cover for the knapsack in X, i.e., C is a subset of N satisfying $\lambda := \sum_{j \in C} a_j - b > 0$ (usually covers for flow problems are called *flow covers*). From $\sum_{j \in N} y_j \leq b$ we obtain

$$\sum_{j \in C} a_j x_j - \sum_{j \in C} s_j \leq b,$$

by discarding all y_j for $j \in N \backslash C$ and replacing y_j by $a_j x_j - s_j$ for all $j \in C$, where $s_j \geq 0$ is a slack variable. Using the mixed knapsack inequality (25), we have that the following inequality is valid for X:

$$\sum_{j \in C} \min(a_j, \lambda) x_j - \sum_{j \in C} s_j \leq \sum_{j \in C} \min(a_j, \lambda) - \lambda,$$

or equivalently, substituting $a_j x_j - y_j$ for s_j,

$$\sum_{j \in C} (y_j + \max(a_j - \lambda, 0)(1 - x_j)) \leq b. \tag{27}$$

It was shown by Padberg, Van Roy, and Wolsey (1985) that this last inequality, called *flow cover inequality*, defines a facet of conv(X), if $\max_{j \in C} a_j > \lambda$.

Flow models have been extensively studied in the literature. Various generalizations of the flow cover inequality (27) have been derived for more complex flow models. In Van Roy and Wolsey (1986), a family of flow cover inequalities is described for a general single node flow model containing variable lower and upper bounds. Generalizations of flow cover inequalities to lot-sizing and capacitated facility location problems can also be found in Aardal, Pochet, and Wolsey (1995) and Pochet (1998). Flow cover inequalities have been used successfully in general purpose branch-and-cut algorithms to tighten formulations of mixed integer sets (Atamtürk, 2002; Gu et al., 1999, 2000; Van Roy and Wolsey, 1987).

3.1.7 Set packing inequalities

The study of set packing polyhedra plays a prominent role in combinatorial optimization and integer programming. Suppose we are given a set $X := \{1, \ldots, m\}$ and a finite system of subsets $X_1, \ldots, X_n \subseteq X$. For each j we have a real number c_j representing the gain for the use of X_j. In the *set packing problem* we ask for a selection $N \subseteq \{1, \ldots, n\}$ such that $X_i \cap X_j = \emptyset$ for all $i, j \in N$ with $i \neq j$ and $\sum_{j \in N} c_j$ is maximal. We can model this problem by introducing incidence vectors $a_j \in \{0, 1\}^m$ for each X_j, $j \in \{1, \ldots, n\}$, where $a_{ij} = 1$ if and only if $i \in X_j$. This defines a matrix $A := (a_{ij}) \in \{0,1\}^{m \times n}$. For the decision which subset we put into the selection N we introduce a vector $x \in \{0,1\}^n$, with $x_j = 1$ if and only if $j \in N$. With this definition we can state the set packing problem as the flowing 0–1 integer program:

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.} & Ax \leq \mathbb{1} \\ & x \in \{0, 1\}^n. \end{array} \tag{28}$$

This problem is important not only from a theoretical but from a computational point of view: set packing problems often occur as subproblems in (mixed) integer problems. Hence a good understanding of 0–1 integer

programs with 0–1 matrices can substantially speed up the solution process of general mixed integer problems including such substructures.

In the sequel we study the *set packing polytope* $P(A) := \text{conv}\{x \in \{0,1\}^n : Ax \leq \mathbb{1}\}$ associated to A. An interpretation of this problem in a graph theoretic sense is helpful to obtain new valid inequalities that strengthens the LP relaxation of (28). The *column intersection graph* $G(A) = (V, E)$ of $A \in \{0,1\}^{m \times n}$ consists of n nodes, one for each column with edges (i, j) between two nodes i and j if and only if their corresponding columns in A have a common nonzero entry in some row. There is a one-to-one correspondence between 0–1 feasible solutions and stable sets in $G(A)$, where a stable set S is a subset of nodes such that $(i, j) \notin E$ for all $i, j \in S$. Consider a feasible vector $x \in \{0,1\}^n$ with $Ax \leq \mathbb{1}$, then $S = \{i \in N : x_i = 1\}$ is a stable set in $G(A)$ and vice versa, each stable set in $G(A)$ defines a feasible 0–1 solution x via $x_i = 1$ if and only if $i \in S$. Observe that different matrices A, A' have the same associated polyhedron if and only if their corresponding intersection graphs coincide. It is therefore customary to study $P(A)$ via the graph G and denote the set packing polytope and the stable set polytope, respectively, by $P(G)$. Without loss of generality we can assume that G is connected.

What can we say about $P(G)$? The following observations are immediate:

(i) $P(G)$ is full dimensional.
(ii) $P(G)$ is lower monotone, i.e., if $x \in P(G)$ and $y \in \{0,1\}^n$ with $0 \leq y \leq x$ then $y \in P(G)$.
(iii) The nonnegativity constraints $x_j \geq 0$ induce facets of $P(G)$.

It is a well-known fact that $P(G)$ is completely described by the nonnegative constraints (iii) and the edge-inequalities $x_i + x_j \leq 1$ for $(i, j) \in E$ if and only if G is bipartite, i.e., there exists a partition (V_1, V_2) of the nodes V such that every edge has one node in V_1 and one in V_2. If G is not bipartite, then it contains odd cycles. They give rise to the following *odd cycle inequality*

$$\sum_{j \in V_C} x_j \leq \frac{|V_C| - 1}{2},$$

where $V_C \subseteq V$ is the set of nodes of cycle $C \subset E$ of odd cardinality. This inequality is valid for $P(G)$ and defines a facet of $P((V_C, E_{V_C}))$ if and only if C is an odd hole, i.e., an odd cycle without chords (Padberg, 1973). This class of inequalities can be separated in polynomial time using an algorithm based on the computation of shortest paths, see Lemma 9.1.11 in Grötschel, Lovász, and Schrijver (1988) for details.

A *clique* (C, E_C) in a graph $G = (V, E)$ is a subset of nodes and edges such that for every pair $i, j \in C, i \neq j$ there exists an edge $(i, j) \in E_C$. From a clique (C, E_C) we obtain the *clique inequality*

$$\sum_{j \in C} x_j \leq 1,$$

which is valid for $P(G)$. It defines a facet of $P(G)$ if and only if the clique is maximal (Fulkerson, 1971; Padberg, 1973). A clique (C, E_C) is said to be *maximal* if every $i \in V$ with $(i,j) \in E$ for all $j \in C$ is already contained in C. In contrast to the class of odd cycle inequalities, the separation of clique inequalities is difficult ($\mathcal{NP}$-hard), see Theorem 9.2.9 in Grötschel, Lovász, and Schrijver (1988). But there exists a larger class of inequalities, called *orthonormal representation* (*OR*) *inequalities*, that includes the clique inequalities and can be separated in polynomial time (Grötschel et al., 1988). Beside odd cycle, clique and OR-inequalities there are many other inequalities known for the stable set polytope. Among these are blossom, odd antihole, and web, wedge inequalities and many more. Borndörfer (1998) gives a survey on these constraints including a discussion on their separability.

3.1.8 Lifted inequalities

The lifting technique is a general approach that has been used in a wide variety of contexts to strengthen valid inequalities. A field for its application is the reuse of inequalities within branch-and-bound, see Section 4, where some inequality that is only valid under certain variable fixings is made globally valid by applying lifting. Assume for simplicity that all integer variables are 0–1. Consider an arbitrary polytope $P \subseteq [0,1]^N$ and let $L \subset N$. Suppose we have an inequality

$$\sum_{j \in L} w_j x_j \leq w_0, \tag{29}$$

which is valid for $P_L := \text{conv}(P \cap \{x : x_j = 0 \ \forall \ j \in N \backslash L\})$. We investigate the lifting of a variable x_j that has been set to 0, setting x_j to 1 is similar. The *lifting problem* is to find lifting coefficients w_j for $j \in N \backslash L$ such that

$$\sum_{j \in N} w_j x_j \leq w_0 \tag{30}$$

is valid for P. Ideally we would like inequality (3) to be "strong," i.e., if inequality (29) defines a face of high dimension of P_L, we would like the inequality (30) to define a face of high dimension of P as well.

One way of obtaining coefficients $(w_j)_{j \in N \backslash L}$ is to apply *sequential lifting:* lifting coefficients w_j are calculated one after another. That is we determine an ordering of the elements of $N \backslash L$ that we follow in computing the coefficients. Let $k \in N \backslash L$ be the first index in this sequence. The coefficient w_k is computed for a given $k \in N \backslash L$ so that

$$w_k x_k + \sum_{j \in L} w_j x_j \leq w_0 \tag{31}$$

is valid for $P_{L \cup \{k\}}$.

We explain the main idea of lifting on the knapsack polytope: $P := P_K(N, a, \beta)$. It is easily extended to more general cases. Define the *lifting function* as the solution of the following 0–1 knapsack problem:

$$\begin{aligned} \Phi_L(u) := \min \quad & w_0 - \sum_{j \in L} w_j x_j \\ \text{s.t.} \quad & \sum_{j \in L} a_j x_j \leq \beta - u, \\ & x \in \{0,1\}^L. \end{aligned}$$

We set $\Phi_L(u) := +\infty$ if $\{x \in \{0,1\}^L : \sum_{j \in L} a_j x_j \leq \beta - u\} = \emptyset$. Then inequality (31) is valid for $P_{L \cup \{k\}}$ if $w_k \leq \Phi_L(a_k)$, see Padberg (1975), Wolsey (1975). Moreover, if $w_k = \Phi_L(a_k)$ and (29) defines a face of dimension t of P_L, then (31) defines a face of $P_{L \cup \{k\}}$ of dimension at least $t+1$.

If one now intends to lift a second variable, then it becomes necessary to update the function Φ_L. Specifically, if $k \in N \backslash L$ was introduced first with a lifting coefficient w_k, then the lifting function becomes

$$\begin{aligned} \Phi_{L \cup \{k\}}(u) := \min \quad & w_0 - \sum_{j \in L \cup \{k\}} w_j x_j \\ \text{s.t.} \quad & \sum_{j \in L \cup \{k\}} a_j x_j \leq \beta - u, \\ & x \in \{0,1\}^{L \cup \{k\}}, \end{aligned}$$

so in general for fixed u, function Φ_L can decrease as more variables are lifted in. As a consequence, lifting coefficients depend on the order in which variables are lifted and therefore different orders of lifting often lead to different valid inequalities.

One of the key questions to be dealt with when implementing such a lifting approach is how to compute lifting coefficients w_j. To perform "exact" sequential lifting (i.e., to compute at each step the lifting coefficient given by the lifting function), we have to solve a sequence of integer programs. In the case of the lifting of variables for the 0–1 knapsack set this can be done efficiently using a dynamic programming approach based on the following recursion formula:

$$\Phi_{L \cup \{k\}}(u) = \min(\Phi_L(u), \Phi_L(u + a_k) - \Phi_L(a_k)).$$

Using such a lifting approach, facet-defining inequalities for the 0–1 knapsack polytope have been derived (Balas, 1975; Balas and Zemel, 1978; Hammer, Johnson, and Peled, 1975; Padberg, 1975; Wolsey, 1975) and embedded in a branch-and-bound framework to solve particular types of 0–1 integer programs to optimality (Crowder et al., 1983).

We now take a look on how to apply the idea of lifting to the more complex polytope associated to the flow problem discussed in Section 3.1.6. Consider the set

$$X' = \left\{ (x, y) \in \{0, 1\}^{L \cup \{k\}} \times \mathbb{R}_+^{L \cup \{k\}} : \sum_{j \in L \cup \{k\}} y_j \leq b, y_j \leq a_j x_j, j \in L \cup \{k\} \right\}.$$

Note that with $(x_k, y_k) = (0, 0)$, this reduces to the flow set, see (26)

$$X = \left\{ (x, y) \in \{0, 1\}^{L} \times \mathbb{R}_+^{L} : \sum_{j \in L} y_j \leq b, y_j \leq a_j x_j, j \in L \right\}.$$

Now suppose that the inequality

$$\sum_{j \in L} w_j x_j + \sum_{j \in L} v_j y_j \leq w_0$$

is valid and facet-defining for $\text{conv}(X)$. As before, let

$$\begin{aligned} \Psi_L(u) = \min \quad & w_0 - \sum_{j \in L} w_j x_j - \sum_{j \in L} v_j y_j \\ \text{s.t.} \quad & \sum_{j \in L} y_j \leq b - u \\ & y_j \leq a_j x_j, \; j \in L \\ & (x, y) \in \{0, 1\}^L \times \mathbb{R}_+^L. \end{aligned}$$

Now the inequality

$$\sum_{j \in L} w_j x_j - \sum_{j \in L} v_j y_j + w_k x_k + v_k y_k \leq w_0$$

is valid for $\text{conv}(X')$ if and only if $w_k + v_k u \leq \Psi_L(u)$ for all $0 \leq u \leq a_k$, ensuring that all feasible points with $(x_k, y_k) = (1, u)$ satisfy the inequality.

The inequality defines a facet if the affine function $w_k + v_k u$ lies below the function $\Psi_L(u)$ in the interval $[0, a_k]$ and touches it in two points different from $(0, 0)$, thereby increasing the number of affinely independent tight points by the number of new variables. In theory, "exact" sequential lifting can be applied to derive valid inequalities for any kind of mixed integer set. However, in practice, this approach is only useful to generate valid inequalities for sets for which one can associate a lifting function that can be evaluated efficiently.

Gu et al. (1999) showed how to lift the pair (x_k, y_k) when y_k has been fixed to a_k and x_k to 1.

Lifting is applied in the context of set packing problems to obtain facets from odd-hole inequalities (Padberg, 1973). Other uses of sequential lifting can be found in Ceria et al. (1998) where the lifting of continuous and integer variables is used to extend the class of lifted cover inequalities to a mixed knapsack set with general integer variables. In Martin (1998), Martin and Weismantel (1998) lifting is applied to define (lifted) feasible set inequalities for an integer set defined by multiple integer knapsack constraints. Generalizations of the lifting procedure where more than one variable is lifted simultaneously (so-called *sequence-independent lifting*) can be found for instance in Atamtürk (2001) and Gu et al. (2000).

3.2 Further relaxations

In the preceding section we have simplified the mixed integer program by relaxing the integrality constraints and by trying to force the integrality of the solution by adding cutting planes. In the methods we are going to discuss now we keep the integrality constraints, but relax part of the constraint matrix that causes difficulties.

3.2.1 Lagrangean relaxation

Consider again (1). The idea of Lagrangean relaxation is to delete part of the constraints and reintroduce them into the problem by putting them into the objective function attached with some penalties. Split A and b into two parts

$$A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix},$$

where $A_1 \in \mathbb{Q}^{m_1 \times n}$, $A_2 \in \mathbb{Q}^{m_2 \times n}$, $b_1 \in \mathbb{Q}^{m_1}$, $b_2 \in \mathbb{Q}^{m_2}$ with $m_1 + m_2 = m$. Then, assuming all equality constraints are divided into two inequalities each, (1) takes the form

$$\begin{aligned} z_{\text{MIP}} := \min \quad & c^T x \\ \text{s.t.} \quad & A_1 x \le b_1 \\ & A_2 x \le b_2 \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}. \end{aligned} \tag{32}$$

Consider for some fixed $\lambda \in \mathbb{R}^{m_1}_+$ the following function

$$\begin{aligned} L(\lambda) = \min \quad & c^T x - \lambda^T (b_1 - A_1 x) \\ \text{s.t.} \quad & x \in P^2, \end{aligned} \tag{33}$$

where $P^2 = \{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} : A_2 x \leq b_2\}$. $L(\cdot)$ is called the *Lagrangean function*. The evaluation of this function for a given λ is called the *Lagrangean subproblem*. Obviously, $L(\lambda)$ is a lower bound on z_{MIP}, since for any feasible solution $\bar{x}$ of (32) we have

$$c^T\bar{x} \geq c^T\bar{x} - \lambda^T(b_1 - A_1\bar{x}) \geq \min_{x \in P^2} c^T x - \lambda^T(b_1 - A_1 x) = L(\lambda).$$

Since this holds for each $\lambda \geq 0$ we conclude that

$$\max_{\lambda \geq 0} L(\lambda) \tag{34}$$

yields a lower bound of z_{MIP}. (34) is called *Lagrangean dual*. Let λ^* be an optimal solution to (34). The questions remain, how good is $L(\lambda^*)$ and how to compute λ^*. The following equation provides an answer to the first question:

$$L(\lambda^*) = \min\{c^T x : A_1 x \leq b_1, x \in \text{conv}(P^2)\}. \tag{35}$$

A proof of this result can be found for instance in Nemhauser and Wolsey (1988) and Schrijver (1986). Since

$$\begin{aligned}\{x \in \mathbb{R}^n : Ax \leq b\} &\supseteq \{x \in \mathbb{R}^n : A_1 x \leq b_1, x \in \text{conv}(P^2)\} \\ &\supseteq \text{conv}\{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} : Ax \leq b\}\end{aligned}$$

we conclude from (35) that

$$z_{\text{LP}} \leq L(\lambda^*) \leq z_{\text{MIP}}. \tag{36}$$

Furthermore, $z_{\text{LP}} = L(\lambda^*)$ for all objective functions $c \in \mathbb{R}^n$ if

$$\{x \in \mathbb{R}^n : A_2 x \leq b_2\} = \text{conv}\{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} : A_2 x \leq b_2\}.$$

It remains to discuss how to compute $L(\lambda^*)$. From a theoretical point of view it can be shown using the polynomial equivalence of separation and optimization that $L(\lambda^*)$ can be determined in polynomial time, if $\min\{\tilde{c}^T x : x \in \text{conv}(P^2)\}$ can be computed in polynomial time for any objective function $\tilde{c}$, see for instance (Schrijver, 1986). In practice, $L(\lambda^*)$ is determined by applying subgradient methods. The function $L(\lambda)$ is piecewise linear, concave and bounded from above. Consider for some fixed $\lambda^0 \in \mathbb{R}_+^{m_1}$ an optimal solution x^0 for (33). Then, $g^0 := A_1 x^0 - b_1$ is a *subgradient* for L and λ^0, i.e.,

$$L(\lambda) - L(\lambda^0) \leq (g^0)^T(\lambda - \lambda^0),$$

since

$$\begin{aligned} L(\lambda) - L(\lambda^0) &= c^T x^\lambda - \lambda^T(b_1 - A_1 x^\lambda) - (c^T x^0 - (\lambda^0)^T(b_1 - A_1 x^0)) \\ &\leq c^T x^0 - \lambda^T(b_1 - A_1 x^0) - (c^T x^0 - (\lambda^0)^T(b_1 - A_1 x^0)) \\ &= (g^0)^T(\lambda - \lambda^0). \end{aligned}$$

Hence, for λ^* we have $(g^0)^T(\lambda^* - \lambda^0) \geq L(\lambda^*) - L(\lambda^0) \geq 0$. In order to find λ^* this suggests to start with some λ^0, compute

$$x^0 \in \operatorname{argmin}\{c^T x - (\lambda^0)^T(b_1 - A_1 x) : x \in P^2\}$$

and determine iteratively $\lambda^0, \lambda^1, \lambda^2, \ldots$ by setting $\lambda^{k+1} = \lambda^k + \mu^k g^k$, where $g^k := A_1 x^k - b_1$, and μ^k is some step length to be specified. This iterative method is the essence of the *subgradient method*. Details and refinements of this method can be found among others in Nemhauser and Wolsey (1988) and Zhao and Luh (2002).

Of course, the quality of the Lagrangean relaxation strongly depends on the set of constraints that is relaxed. On one side, we must compute (33) for various values of λ and thus it is necessary to compute $L(\lambda)$ fast. Therefore one may want to relax as many (complicated) constraints as possible. On the other hand, the more constraints are relaxed the worse the bound $L(\lambda^*)$ will get, see Lemaréchal and Renaud (2001). Therefore, one always must find a compromise between these two conflicting goals.

3.2.2 Dantzig–Wolfe decomposition

The idea of decomposition methods is to decouple a set of constraints (variables) from the problem and treat them at a superordinate level, often called the *master problem*. The resulting residual subordinate problem can often 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. In this section we discuss two well known examples of this approach, Dantzig–Wolfe decomposition and Benders' decomposition. We will see that as in the case of Lagrangean relaxation these methods also delete part of the constraint matrix. But instead of reintroducing this part in the objective function, it is now reformulated and reintroduced into the constraint system.

Let us start with Dantzig–Wolfe decomposition (Dantzig and Wolfe, 1960) and consider again (32), where we assume for the moment that $p = 0$, i.e., a linear programming problem. Consider the polyhedron $P^2 = \{x \in \mathbb{R}^n : A_2 x \leq b_2\}$. It is a well known fact about polyhedra that there exist vectors $v_1, \ldots, v_k$ and $e_1, \ldots, e_l$ such that $P^2 = \operatorname{conv}(\{v_1, \ldots, v_k\}) + \operatorname{cone}(\{e_1, \ldots, e_l\})$. In other words, $x \in P^2$ can be written in the form

$$x = \sum_{i=1}^{k} \lambda_i v_i + \sum_{j=1}^{l} \mu_j e_j \tag{37}$$

with $\lambda_1, \ldots, \lambda_k \geq 0$, $\sum_{i=1}^{k} \lambda_i = 1$ and $\mu_1, \ldots, \mu_l \geq 0$. Substituting for x from (37) we may write (32) as

$$\begin{aligned} \min \quad & c^T\left(\sum_{i=1}^{k}\lambda_i v_i + \sum_{j=1}^{l}\mu_j e_j\right) \\ \text{s.t.} \quad & A_1\left(\sum_{i=1}^{k}\lambda_i v_i + \sum_{j=1}^{l}\mu_j e_j\right) \leq b_1 \\ & \sum_{i=1}^{k}\lambda_i = 1 \\ & \lambda \in \mathbb{R}^k_+, \mu \in \mathbb{R}^l_+, \end{aligned}$$

which is equivalent to

$$\begin{aligned} \min \quad & \sum_{i=1}^{k}(c^T v_i)\lambda_i + \sum_{j=1}^{l}(c^T e_j)\mu_j \\ \text{s.t.} \quad & \sum_{i=1}^{k}(A_1 v_i)\lambda_i + \sum_{j=1}^{l}(A_1 e_j) \leq b_1 \\ & \sum_{i=1}^{k}\lambda_i = 1 \\ & \lambda \in \mathbb{R}^k_+, \mu \in \mathbb{R}^l_+. \end{aligned} \tag{38}$$

Problem (38) is called the *master problem* of (32). Comparing formulations (32) and (38) we see that we reduced the number of constraints from m to m_1, but obtained $k+l$ variables instead of n. $k+l$ might be large compared to n, in fact even exponential (consider for example the unit cube in $\mathbb{R}^n$ with $2n$ constraints and 2^n vertices) so that there seems to be at first sight no gain in using formulation (38). However, we can use the simplex algorithm for the solution of (38). For ease of exposition abbreviate (38) by $\min\{w^T\eta : D_\eta = d, \eta \geq 0\}$ with $D \in \mathbb{R}^{(m_1+1)\times(k+l)}$, $d \in \mathbb{R}^{m_1+1}$. Recall that the simplex algorithm starts with a (feasible) basis $B \subseteq \{1, \ldots, k+l\}$, $|B| = m_1 + 1$, with D_B nonsingular and the corresponding (feasible) solution $\eta^*_B = D_B^{-1}d$ and $\eta^*_N = 0$, where $N = \{1, \ldots, k+l\} \backslash B$. Observe that $D_B \in \mathbb{R}^{(m_1+1)\times(m_1+1)}$ is (much) smaller than a basis for the original system (32) and that only a fraction of variables ($m_1 + 1$ out of $k+l$) are possibly nonzero. In addition, on the way to an optimal solution the only operation within the simplex method that involves all columns is the pricing step, where it is checked whether the reduced costs $w_N - \tilde{y}^T D_N$ are nonnegative with $\tilde{y} \leq 0$ being the solution of $y^T D_B = w_B$. The nonnegativity of the reduced costs can be verified via the

following linear program:

$$\begin{array}{ll} \min & (c^T - \bar{y}^T A_1)x \\ \text{s.t.} & A_2 x \leq b_2 \\ & x \in \mathbb{R}^n, \end{array} \tag{39}$$

where $\bar{y}$ are the first m_1 components of the solution of $\tilde{y}$. The following cases might come up:

(i) Problem (39) has an optimal solution $\tilde{x}$ with $(c^T - \bar{y}^T A_1)\tilde{x} < \tilde{y}_{m_1+1}$. In this case, $\tilde{x}$ is one of the vectors v_i, $i \in \{1, \ldots, k\}$, with corresponding reduced cost

$$w_i - \tilde{y}^T D_{\cdot i} = c^T v_i - \tilde{y}^T \begin{pmatrix} A_1 v_i \\ 1 \end{pmatrix} = c^T v_i - \bar{y}^T A_1 v_i - \tilde{y}_{m_1+1} < 0.$$

In other words, $\binom{A_1 v_i}{1}$ is the entering column within the simplex algorithm.

(ii) Problem (39) is unbounded.
Here we obtain a feasible extreme ray e^* with $(c^T - \bar{y}^T A_1)e^* < 0$. e^* is one of the vectors e_j, $j \in \{1, \ldots, l\}$. It yields a column $\binom{A_1 e_j}{0}$ with reduced cost

$$w_{k+j} - D_{\cdot(k+j)} = c^T e_j - \tilde{y}^T \begin{pmatrix} A_1 e_j \\ 0 \end{pmatrix} = c^T e_j - \bar{y}^T (A_1 e_j) < 0.$$

That is, $\binom{A_1 e_j}{0}$ is the entering column.

(iii) Problem (39) has an optimal solution $\tilde{x}$ with $(c^T - \bar{y}^T A_1)^T \tilde{x} \geq \tilde{y}_{m_1+1}$. In this case we conclude using the same arguments as in (i) and (ii) that $w_i - \tilde{y}^T D_{\cdot i} \geq 0$ for all $i = 1, \ldots, k+l$ proving that x^* is an optimal solution for the master problem (38).

Observe that the whole problem (32) is decomposed into two problems, i.e., (38) and (39), and the approach iteratively works on the master level (38) and the subordinate level (39). The procedure starts with some feasible solution for (38) and generates new promising columns on demand by solving (39). Such procedures are commonly called *column generation* or *delayed column generation algorithms*.

The approach can also be extended to general integer programs with some caution. In this case problem (39) turns from a linear to an integer linear program. In addition, we have to guarantee in (37) that all feasible integer solutions x of (32) can be generated by (integer) linear combinations of the vectors $v_1, \ldots, v_k$ and $e_1, \ldots, e_l$, where

$$\operatorname{conv}(\{x \in \mathbb{Z}^n : Ax \leq b\}) = \operatorname{conv}(\{v_1, \ldots, v_k\}) + \operatorname{cone}(\{e_1, \ldots, e_l\}).$$

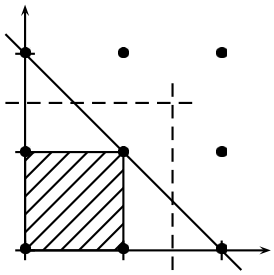

Fig. 2. Extending Dantzig–Wolfe decomposition to integer programs.

It is not sufficient to require λ and μ to be integer. Consider as a counterexample

$$A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ b_1 = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix} \quad \text{and} \quad A_2 = (1, 1),\ b_2 = 2$$

and the problem

$$\max\{x_1 + x_2 : A_1 x \leq b_1, A_2 x \leq b_2, x \in \{0, 1, 2\}^2\}$$

Then

$$P^2 = \operatorname{conv}\left(\left\{\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \end{pmatrix}\right\}\right),$$

see Fig. 2, but the optimal solution $\binom{1}{1}$ of the integer program is not an integer linear combination of the vertices of P^2. However, when all variables are 0–1, this difficulty does not occur, since any 0–1 solution of the LP relaxation of some binary MIP is always a vertex of that polyhedron. And in fact, column generation algorithms are not only used for the solution of large linear programs, but especially for large 0–1 integer programs. Of course, the Dantzig–Wolfe decomposition for linear or 0–1 integer programs is just one type of column generation algorithm. Others solve the subordinate problem not via general linear or integer programming techniques, but use combinatorial or explicit enumeration algorithms. Furthermore, the problem is often not modeled via (32), but directly as in (38). This is, for instance, the case when the set of feasible solutions have a rather complex description by linear inequalities, but these constraints can easily be incorporated into some enumeration scheme.

3.2.3 Benders' decomposition

Let us finally turn to *Benders' decomposition* (Benders, 1962). Benders' decomposition also deletes part of the constraint matrix, but in contrast to

Dantzig–Wolfe decomposition, where we delete part of the constraints and reintroduce them via column generation, we now delete part of the variables and reintroduce them via cutting planes. In this respect, Benders' decomposition is the same as Dantzig–Wolfe decomposition applied to the dual as we will see in detail in Section 3.2.4. Consider again (1) and write it in the form

$$\begin{array}{ll} \min & c_1^T x_1 + c_2^T x_2 \\ \text{s.t.} & A_1 x_1 + A_2 x_2 \leq b \\ & x_1 \in \mathbb{R}^{n_1}, x_2 \in \mathbb{R}^{n_2}, \end{array} \tag{40}$$

where $A = [A_1,\ A_2] \in \mathbb{R}^{m \times n}$, $A_1 \in \mathbb{R}^{m \times n_1}$, $A_2 \in \mathbb{R}^{m \times n_2}$, $c_1,\ x_1 \in \mathbb{R}^{n_1}$, $c_2,\ x_2 \in \mathbb{R}^{n_2}$ with $n_1 + n_2 = n$. Note that we have assumed for ease of exposition the case of a linear program. We will see, however, that what follows is still true if $x_1 \in \mathbb{Z}^{n_1}$. Our intention is to get rid of the variables x_2. These variables prevent (40) from being a pure integer program in case $x_1 \in \mathbb{Z}^{n_1}$. Also in the linear programming case they might be the origin for some difficulties, see the applications in Section 4.4. One well known approach to get rid of variables is projection, see also the lift-and-project cuts in Section 3.1. In order to apply projection we must slightly reformulate (40) to

$$\begin{array}{ll} \min & z \\ \text{s.t.} & -z + c_1^T x_1 + c_2^T x_2 \ \leq 0 \\ & \qquad A_1 x_1 + A_2 x_2 \ \leq b \\ & z \in \mathbb{R}, x_1 \in \mathbb{R}^{n_1}, x_2 \in \mathbb{R}^{n_2}. \end{array} \tag{41}$$

Now, (41) is equivalent to

$$\begin{array}{ll} \min & z \\ \text{s.t.} & -uz + uc_1^T x_1 + v^T A_1 x_1 \leq v^T b \\ & z \in \mathbb{R}, x_1 \in \mathbb{R}^{n_1}, \\ & \begin{pmatrix} u \\ v \end{pmatrix} \in C, \end{array} \tag{42}$$

where

$$C = \left\{ \begin{pmatrix} u \\ v \end{pmatrix} \in \mathbb{R}^{m+1} : v^T A_2 + u c_2^T = 0, u \geq 0, v \geq 0 \right\}.$$

C is a pointed polyhedral cone, thus there exist vectors

$$\begin{pmatrix} \bar{u}_1 \\ \bar{v}_s \end{pmatrix}, \ldots, \begin{pmatrix} \bar{u}_s \\ \bar{v}_s \end{pmatrix}$$

such that

$$C = \operatorname{cone}\left(\left\{\begin{pmatrix}\bar{u}_1\\ \bar{v}_1\end{pmatrix}, \ldots, \begin{pmatrix}\bar{u}_s\\ \bar{v}_s\end{pmatrix}\right\}\right).$$

These extreme rays can be rescaled such that $\bar{u}_i$ is zero or one. Thus

$$C = \operatorname{cone}\left(\left\{\begin{pmatrix}0\\ v_k\end{pmatrix} : k \in K\right\}\right) + \operatorname{cone}\left(\left\{\begin{pmatrix}1\\ v_j\end{pmatrix} : j \in J\right\}\right)$$

with $K \cup J = \{1, \ldots, s\}$ and $K \cap J = \emptyset$. With this description of C, (42) can be restated as

$$\begin{array}{lllll} \min & z & & & \\ \text{s.t.} & -z \le c_1^T x_1 + v_j^T(b - A_1 x_1) & \text{for all} & j \in J, \\ & 0 \le \qquad\quad v_k^T(b - A_1 x_1) & \text{for all} & k \in K, \\ & z \in \mathbb{R}, x_1 \in \mathbb{R}^{n_1}. & & \end{array} \tag{43}$$

Problem (43) is called *Benders' master problem*. Benders' master problem has just $n_1 + 1$ variables instead of $n_1 + n_2$ variables in (40), or in case $x_1 \in \mathbb{Z}^{n_1}$ we have reduced the mixed integer program (40) to an almost pure integer program (43) with one additional continuous variable z. However, (43) contains an enormous number of constraints, in general exponentially many in n. To get around this problem, we solve Benders' master problem by cutting plane methods, see Section 3.1. We start with a small subset of extreme rays of C (possibly the empty set) and optimize (43) just over this subset. We obtain an optimal solution x^*, z^* of the relaxed problem and we must check whether this solution satisfies all other inequalities in (43). This can be done via the following linear program

$$\begin{array}{ll} \min & v^T(b - A_1 x_1^*) + u(z^* - c_1^T x_1^*) \\ \text{s.t.} & \begin{pmatrix}u\\ v\end{pmatrix} \in C. \end{array} \tag{44}$$

Problem (44) is called the *Benders' subproblem*. It is feasible, since $\binom{0}{0} \in C$, and (44) has an optimal solution value of zero or it is unbounded. In the first case, x_1^*, z^* satisfies all inequalities in (43) and we have solved (43) and thus (40). In the latter case we obtain an extreme ray $\binom{u^*}{v^*}$ from (44) with $(v^*)^T(b - A_1 x_1^*) + u^*(z^* - c_1^T x_1^*) < 0$ which after rescaling yields a cut for (43) violated by x_1^*, z^*. We add this cut to Benders' master problem (43) and iterate.

3.2.4 Connections between the approaches

At first sight, Lagrangean relaxation, Dantzig–Wolfe, and Benders' decomposition seem to be completely different relaxation approaches. However, they are strongly related as we will shortly outline in the following. Consider once again (39) which for some fixed $\bar{y} \leq 0$ can be rewritten as

$$\begin{array}{lll} \min & (c^T - \bar{y}^T A_1)x = \min & c^T x + \bar{y}^T(b_1 - A_1 x) - \bar{y}^T b_1 \\ \text{s.t.} & x \in P^2 & \text{s.t.} \quad x \in P^2 \\ & & = L(-\bar{y}) - \bar{y}^T b, \end{array}$$

that is, (33) and (39) are the same problems up to the constant $-\bar{y}^T b$. Even further, by replacing P^2 by $\text{conv}(\{v_1, \ldots, v_k\}) + \text{cone}(\{e_1, \ldots, e_l\})$ we see that (38) coincides with the right-hand side in (35) and thus with $L(\lambda^*)$. In other words, both Dantzig–Wolfe and Lagrangean relaxation compute the same bound. The only differences are that for updating the dual variables, i.e., λ in the Lagrangean relaxation and $\bar{y}$ in Dantzig–Wolfe, in the first case subgradient methods whereas in the latter linear programming techniques are applied. Other ways to compute λ^* are provided by the bundle method based on quadratic programming (Hiriart-Urruty and Lemarechal, 1993), and the analytic center cutting plane method that is based on an interior point algorithm (Goffin and Vial, 2002).

Similarly, Benders' decomposition is the same as that applied by Dantzig–Wolfe to the dual of (40). To see this, consider its dual

$$\begin{array}{ll} \max & -y^T b \\ \text{s.t.} & -y^T A_1 = c_1^T \\ & -y^T A_2 = c_2^T \\ & y \geq 0. \end{array} \tag{45}$$

Now reformulate $\bar{P}^2 = \{y \in \mathbb{R}^{n_2} : y^T A_2 = -c_2^T, y \leq 0\}$ by $\bar{P}^2 = \text{conv}(\{v_j : j \in J\}) + \text{cone}(\{v_k : k \in K\})$, where K, J and v_l, $l \in K \cup J$ are exactly those from (43), and rewrite (45) as

$$\begin{array}{ll} \max & \displaystyle\sum_{j \in J}(-v_j^T b)\lambda_j + \sum_{k \in K}(-v_k^T b)\mu_k \\ \text{s.t.} & \displaystyle\sum_{j \in J}(-v_j^T A_1)\lambda_j + \sum_{k \in K}(-v_k^T A_1)\mu_k = c_1^T \\ & \displaystyle\sum_{j \in J}\lambda_j = 1 \\ & \lambda \in \mathbb{R}_+^J, \mu \in \mathbb{R}_+^K. \end{array} \tag{46}$$

Now from Section 3.2.2 we conclude that (46) is the master problem from (45). Finally, dualizing (46) yields

$$\begin{aligned} \min \quad & c_1^T x_1 + z \\ \text{s.t.} \quad & v_i^T (b - A_1 x_1) \geq -z \ \forall j \in J \\ & v_k^T (b - A_1 x_1) \geq 0 \quad \forall k \in K, \end{aligned}$$

which is equivalent to (43), that is to the Benders' master problem of (40). In other words, Benders' and Dantzig–Wolfe decomposition yield the same bound, which by our previous discussion is also equivalent to the Lagrangean dual (34).

4 Branch-and-bound strategies

Branch-and-bound algorithms for mixed integer programming use a "divide and conquer" strategy to explore the set of all feasible mixed integer solutions. But instead of exploring the whole feasible set, they make use of lower and upper bounds and therefore avoid surveying certain (large) parts of the space of feasible solutions. Let $X := \{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} : Ax \leq b\}$ be the set of feasible mixed integer solutions of problem (1). If it is too difficult to compute

$$\begin{aligned} z_{\text{MIP}} = \min \quad & c^T x \\ \text{s.t.} \quad & x \in X \end{aligned}$$

(for instance with a cutting plane approach), we can split X into a finite number of subsets $X_1, \ldots, X_k \subset X$ such that $\cup_{j=1}^{k} X_j = X$ and then try to solve separately each of the subproblems

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & x \in X_j, \quad \forall j = 1, \ldots, k. \end{aligned}$$

Later, we compare the optimal solutions of the subproblems and choose the best one. Each subproblem might be as difficult as the original problem, so one tends to solve them by the same method, i.e., splitting the subproblems again into further sub-subproblems. The (fast-growing) list of all subproblems is usually organized as a tree, called a *branch-and-bound tree*. Since this tree of subproblems looks like a family tree, one usually says that a *father* or *parent* problem is split into two or more *son* or *child* problems. This is the branching part of the branch-and-bound method.

For the bounding part of this method we assume that we can efficiently compute a lower bound b_{X_j} of subproblem X_j, i.e., $b_{X_j} \leq \min_{x \in X_j} c^T x$. In the case of mixed integer programming, this lower bound can be obtained by

using any relaxation method discussed in Section 3. In the following, suppose we have chosen the LP relaxation method by relaxing the integrality constraints. In Section 4.4 we give references if one of the other relaxation methods is applied within branch-and-bound. For the ease of explanation we assume in the sequel that the LP relaxation has a finite optimum. It occasionally happens in the course of the branch-and-bound algorithm that the optimal solution $\tilde{x}_{X_j}$ of the LP relaxation of a subproblem X_j is also a feasible mixed integer point, i.e., it lies in X. This allows us to maintain an upper bound $U := c^T \tilde{x}_{X_j}$ on the optimal solution value z_{MIP} of X, as $z_{\mathrm{MIP}} \leq U$. Having a good upper bound U is crucial in a branch-and-bound algorithm, because it keeps the branching tree small: suppose the solution of the LP relaxation of some other subproblem X_j satisfies $b_{X_j} \geq U$. Then subproblem X_j and further sub-subproblems derived from X_j need not be considered further, because the optimal solution of this subproblem cannot be better than the best feasible solution $\tilde{x}_{X_j}$ corresponding to U. The following algorithm summarizes the whole procedure:

Algorithm 5. (*Branch-and-bound*)

1. *Let L be the list of unsolved problems. Initialize L with* (1). *Set $U := +\infty$ as upper bound.*
2. *Choose an unsolved problem X_j from the list L and delete it from L.*
3. *Compute the lower bound b_{X_j} by solving the linear programming relaxation. If problem X_j is infeasible, go to Step* 2 *until the list is empty. Otherwise, let $\tilde{x}_{X_j}$ be an optimal solution and set $b_{X_j} := c^T \tilde{x}_{X_j}$.*
4. *If $\tilde{x}_{X_j} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$, problem X_j is solved and we found a feasible solution of X_j; if $U > b_{X_j}$ set $U := b_{X_j}$ and delete all subproblems X_i with $b_{X_i} \geq U$ from the list L.*
5. *If $\tilde{x}_{X_j} \notin \mathbb{Z}^p \times \mathbb{R}^{n-p}$, split problem X_j into subproblems and add them to the list L.*
6. *Go to Step* 2 *until L is empty.*

Each (sub)problem X_j in the list L corresponds to a node in the branch-and-bound tree, where the unsolved problems are the leaves of the tree and the node that corresponds to the entire problem (1) is the root.

As crucial as finding a good upper bound is to find a good lower bound. Sometimes the LP relaxation turns out to be weak, but can be strengthened by adding cutting planes as discussed in Section 3.1. This combination of finding cutting planes and branch-and-bound leads to a hybrid algorithm called a *branch-and-cut algorithm*.

Algorithm 6. (*Branch-and-cut*)

1. *Let L be the list of unsolved problems. Initialize L with* (1). *Set $U := +\infty$ as upper bound.*
2. *Choose an unsolved problem X_j from the list L and delete it from L.*

3. *Compute the lower bound b_{X_j} by solving the linear programming relaxation. If problem X_j is infeasible, go to Step* 2 *until the list is empty. Let $\tilde{x}_{X_j}$ be an optimal solution and set $b_{X_j} := c^T \tilde{x}_{X_j}$.*
4. *If $\tilde{x}_{X_j} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$, problem X_j is solved and we found a feasible solution of X_j; if $U > b_{X_j}$ set $U := b_{X_j}$ and delete all subproblems X_i with $b_{X_i} \geq U$ from the list L.*
5. *If $\tilde{x}_{X_j} \notin \mathbb{Z}^p \times \mathbb{R}^{n-p}$, look for cutting planes and add them to the linear relaxation.*
6. *Go to Step* 3 *until no more violated inequalities can be found or violated inequalities have too little impact on improving the lower bound.*
7. *Split problem X_j into subproblems and add them to the list L.*
8. *Go to Step* 2 *until L is empty.*

In the general outline of the above branch-and-cut algorithm, there are two steps in the branch-and-bound part that leave some choices. In Step 2 of Algorithm 6 we have to select the next problem (node) from the list of unsolved problems to work on next, and in Step 7 we must decide on how to split the problem into subproblems. Usually this split is performed by choosing a variable $\tilde{x}_j \notin \mathbb{Z}$, $1 \leq j \leq p$, from an optimal solution of some subproblem X_k from the list of open problems and creating two subproblems: one with the additional bound $x_j \leq \lfloor \tilde{x}_j \rfloor$ and the other with $x_j \geq \lceil \tilde{x}_j \rceil$. Popular strategies are to branch on a variable that is closest to 0.5 and to choose a node with the worst dual bound, i.e., a problem $\tilde{j}$ from the list of open problems with $b_{X_{\tilde{j}}} = \min_j b_{X_j}$. In this section we briefly discuss some more alternatives that outperform the standard strategies. For a comprehensive study of branch-and-bound strategies we refer to Land and Powell (1979), Linderoth and Savelsbergh (1999), Achterberg, Koch, and Martin (2005), and the references therein.

4.1 Node selection

In this section we discuss three different strategies to select the node to be processed next, see Step 3 of Algorithm 6.

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 (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.

Depth first search (dfs). This rule chooses some 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 one of the two sons is always processed next, if the node can 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.

Best projection. When selecting a node the most important 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 for this rule is a paper of 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, $\bar{z}_{\text{IP}}$ the value of 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

$$\varrho(p) = z(p) + \frac{\bar{z}_{\text{IP}} - z(\text{root})}{s(\text{root})} \cdot s(p). \tag{47}$$

The term $(\bar{z}_{\text{IP}} - z(\text{root})/s(\text{root}))$ 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 $\varrho(\cdot)$.

The computational tests in Martin (1998) show that *dfs* finds by far the largest number of feasible solutions. This indicates that feasible solutions tend to lie deep in the branch-and-bound tree. In addition, the number of simplex iterations per LP is on an average much smaller (around one half) for *dfs* than for *bfs* or *best projection*. This confirms our statement that reoptimizing a linear program is fast when just one variable bound is changed. However, the *dfs* strategy does not take the dual bound into account. For many more difficult problems the dual bound is not improved resulting in very bad solution guarantees compared to the other two strategies. *Best projection* and *bfs* are doing better in this respect. There is no clear winner between the two, sometimes *best projection* outperforms *bfs*, but on average *bfs* is the best. Linderoth and Savelsbergh (1999) compare further node selection strategies and come to a similar conclusion that there is no clear winner and that a sophisticated MIP solver should allow many different options for node selection.

4.2 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 7 of

Algorithm 6. The only way to split a problem within an LP based branch-and-bound algorithm 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 the addition of an inequality, but only a change of the bounds of variable j. Branching on more complicated inequalities or even splitting the problem into more than two subproblems are rarely incorporated into general solvers, but turn out to be effective in special cases, see, for instance, Borndörfer et al. (1998), Clochard and Naddef (1993), Naddef (2002). In the following we present three variable selection rules.

Most infeasibility. This rule is to choose a 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.

Pseudo-costs. This is a more sophisticated rule in the sense that it keeps a history of the success of the variables on which one has already branched. To introduce this rule, which goes back to (Benichou et al., 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.
$\upsilon(p)$ 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

$$\Phi^+(j) = \frac{1}{|P_j^+|} \sum_{p \in P_j^+} \frac{z(p) - z(f(p))}{\lceil x_{\upsilon(p)}(f(p)) \rceil - x_{\upsilon(p)}(f(p))}, \tag{48}$$

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

$$\Phi^-(j) = \frac{1}{|P_j^-|} \sum_{p \in P_j^-} \frac{z(p) - z(f(p))}{x_{v(p)}(f(p)) - \lfloor x_{v(p)}(f(p)) \rfloor}, \tag{49}$$

where $P_j^- \subseteq \mathcal{P}^-$. The terms

$$\frac{z(p) - z(f(p))}{\lceil x_{v(p)}(f(p)) \rceil - x_{v(p)}(f(p))} \quad \text{and} \quad \frac{z(p) - z(f(p))}{x_{v(p)}(f(p)) - \lfloor x_{v(p)}(f(p)) \rfloor},$$

respectively, measure the change in the objective function per unit decrease of infeasibility of variables j. There are many suggestions made on how to choose the sets P_j^+ and P_j^-, for a survey see Linderoth and Savelsbergh (1999). To name one possibility, following the suggestion of Eckstein (1994) one could choose $P_j^+ := \{p \in \mathcal{P}^+ : v(p) = j\}$ and $P_j^- := \{p \in \mathcal{P}^- : v(p) = j\}$, if j has already been considered as a branching variable, otherwise set $P_j^+ := \mathcal{P}^+$ 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 one typically sets

$$\Phi(j) = \alpha_j^+ \Phi^+(j)(\lceil \bar{x}_j \rceil - x_j) + \alpha_j^- \Phi^-(j)(\bar{x}_j - \lfloor \bar{x}_j \rfloor), \tag{50}$$

where α_j^+, α_j^- are positive scalars. A variable that maximizes (50) is chosen to be the next branching variable. As formula (50) shows, the rule takes the earlier 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. An attempt is made to overcome this drawback in the following rule.

Strong branching. The idea of *strong branching*, invented by CPLEX (ILOG CPLEX Division, 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. Needless to say that each MIP solver has its own parameter settings, all are of heuristic nature and that their justifications are based only on experimental results.

Computational experience in Martin (1998) show 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 particularly apparent if the number of already solved branch-and-bound nodes is large. In this case the function $\Phi(\cdot)$ properly represents the variables that are qualified for branching. In addition, the time necessary to compute the *pseudo-costs* is basically for free. The statistics change when looking at *strong branching*. *Strong branching* is much more expensive than the other two strategies. This comes as no surprise, since in general the average number of dual simplex iterations per linear program is very small (for the `Miplib`, for instance, below 10 on average). Thus, the testing of a certain number of variables (even if it is small) in *strong branching* is relatively expensive. On the other hand, the number of branch-and-bound nodes is much smaller (around one half) compared to the *pseudo-costs* strategy. This decrease, however, does not completely compensate the higher running times for selecting the variables in general. Thus, *strong branching* is normally not used as a default strategy, but can be a good choice for some hard instances. A similar report is given in Linderoth and Savelsbergh (1999), where Linderoth and Savelsbergh conclude that there is no branching rule that clearly dominates the others, though pseudo-cost strategies are essential to solve many instances.

The latter strategy is refined in several aspects in Linderoth and Savelsbergh (1999), Achterberg, Koch, and Martin (2005) to hybrid methods where the advantages of pseudo-cost and strong branching are put together. The basic idea is to use strong branching at the very beginning when pseudo-costs contain no or only poor information and switches to the much faster pseudo-cost strategy later in the solution process.

4.3 Further aspects

In this section we discuss some additional issues that can be found in basically every state-of-the-art branch-and-cut implementation.

LP management. The method that is commonly used to solve the LPs within a branch-and-cut algorithm is the dual simplex algorithm, because an LP basis stays dual feasible when adding cutting planes. There are fast and robust linear programming solvers available, see, for instance, DASH Optimization (2001) and ILOG CPLEX Division (2000). Nevertheless, one major aspect in the design of a branch-and-cut algorithm is to control the size of the linear programs. To this end, inequalities are often 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. 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 applications. It is clear that one always makes sure that no redundant inequalities are added to the linear program. A commonly used data structure in this context is the *pool*. Violated inequalities that are added to the LP are stored in this data 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 access 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 fathoming of branches of the search-tree is another. Primal heuristics strongly depend on the application. A very common way to find feasible solutions for general mixed integer programs is to "plunge" from time to time at some node of the branch-and-bound tree, i.e., to dive deeper into the tree and look for feasible solutions. This plunging is done by alternating rounding/fixing some variables and solving linear programs, until all the variables are fixed, the LP is infeasible, a feasible solution has been found, or the LP value exceeds the current best solution. This rounding heuristic can be detached from the regular branch-and-bound enumeration phase or considered within the global enumeration phase. The complexity and the sensitivity to the change of the LP solutions influences the frequency with which the heuristics are called. Some more information on this topic can be found, for instance, in Bixby, Fenelon, Guand, Rothberg, and Wunderling (1998), Cordier, Marchand, Laundy, and Wolsey (1999), Martin (1998).

Some ideas that go beyond this general approach of rounding and fixing variables can be found in Balas, Ceria, Dawande, Margot, and Pataki (2001), Balas and Martin (1980), Fischetti and Lodi (2002). Balas et al. (2001) observe that an LP solution consisting solely of slack variables must be integer and thus try to pivot in slack variables into the optimal LP solution to derive feasible integer solutions. In Balas et al. (2001) 0–1 solutions are generated by doing local search in a more sophisticated manner. Very recently, a new idea was proposed by Fischetti and Lodi (2002). Instead of fixing certain variables, they branch on the constraint that any new solution must have at least or at most a certain number of fixings in common with the current best solution. The computational

results show that with this branching rule very fast good feasible solutions are obtained.

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 solutions, z^{IP} be an upper bound on the value of the optimal solution, and $d = (d_i)_{i=1,\ldots,n}$ the corresponding reduced cost vector. Consider a nonbasic variable x_i of the current LP solution with finite lower and upper bounds l_i and u_i, and nonzero reduced cost d_i. Set $\delta = (z^{\mathrm{IP}} - \bar{z}/|d_i|)$, rounded down in case x_j is a binary or an 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 (Crowder et al., 1983), in which case the variable can always be fixed if the criterion applied. There are problems where by the reduced cost criterion many variables can be fixed, see, for instance, (Ferreira, Martin, and Weismantel, 1996). 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 (i.e., a constraint of the form $\sum_{j \in J} x_j \leq 1$ with nonnegative variables x_j), all other variables in this SOS constraint can be fixed to zero.

4.4 Other relaxation methods within branch-and-bound

We have put our emphasis up to now on branch-and-cut algorithms where we investigated the LP-relaxation in combination with the generation of cutting planes. Of course the bounding within branch-and-bound algorithms could also be obtained by any other relaxation method discussed in Section 3.2.

Dantzig–Wolfe decomposition or delayed column generation in connection with branch-and-bound is commonly called *branch-and-price algorithm*. Branch-and-price algorithms have been successfully applied for instance in airline crew scheduling, vehicle routing, public mass transport, or network design, to name just a few. An outline of recent developments, practical applications, and implementation details of branch-and-price can be found for instance in Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance (1998), Lübbecke and Desrosiers (2002), Savelsbergh (2001), Vanderbeck (1999), (2000). Of course, also integer programs with bordered block diagonal form, see Fig. 1, nicely fit into this context. In contrast to Lagrangean relaxation, see below, where the coupling constraints are relaxed, Dantzing–Wolfe decomposition keeps these constraints in the master problem and relaxes

the constraints of the blocks having the advantage that (39) decomposes into independent problems, one for each block.

Lagrangean relaxation is very often used if the underlying linear programs of (1) are just too big to be solved directly and even the relaxed problems in (33) are still large (Löbel, 1997, 1998). Often the relaxation can be done in a way that the evaluation of (33) can be solved combinatorially. In the following we give some applications where this method has been successfully applied and a good balance between these two opposite objectives can be found.

Consider the traveling salesman problem where we are given a set of nodes $V = \{1, \ldots, n\}$ and a set of edges E. The nodes are the cities and the edges are pairs of cities that are connected. Let $c_{(i,j)}$ for $(i,j) \in E$ denote the traveling time from city i to city j. The traveling salesman problem (TSP) now asks for a tour that starts in city 1, visits every other city exactly once, returns to city 1 and has minimal travel time. We can model this problem by the following 0–1 integer program. The binary variable $x_{(i,j)} \in \{0,1\}$ equals 1 if city j is visited right after city i is left, and equals 0 otherwise, that is $x \in \{0,1\}^E$. The equations

$$\sum_{\{i:(i,j)\in E\}} x_{(i,j)} = 2 \quad \forall j \in V$$

(*degree constraints*) ensure that every city is entered and left exactly once, respectively. To eliminate subtours, for any $U \subset V$ with $2 \le |U| \le |V| - 1$, the constraints

$$\sum_{\{(i,j)\in E: i,j\in U\}} x_{(i,j)} \le |U| - 1$$

have to be added. By relaxing the degree constraints in the integer programming formulation for the traveling salesman problem, we are left with a spanning tree problem, which can be solved fast by the greedy algorithm. A main advantage of this TSP relaxation is that for the evaluation of (33) combinatorial algorithms are at hand and no general LP or IP solution techniques must be used. Held and Karp (1971) proposed this approach in the seventies and they solved instances that could not be solved with any other method at that time.

Other examples where Lagrangean relaxation is used are multicommodity flow problems arising for instance in vehicle scheduling or scenario decompositions of stochastic mixed integer programs. In fact, the latter two applications fall into a class of problems where the underlying matrix has bordered block diagonal form, see Fig. 1. If we relax the coupling constraints within a Lagrangean relaxation, the remaining matrix decomposes into k independent blocks. Thus, $L(\lambda)$ is the sum of k individual terms that can be determined separately. Often each single block A_i models a network flow

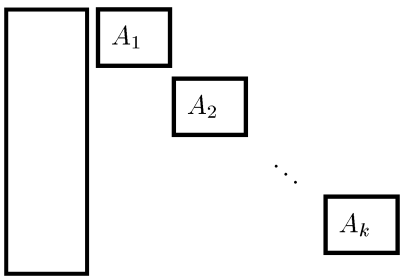

Fig. 3. Matrix in bordered block diagonal form with coupling variables.

problem, a knapsack problem or the like and can thus be solved using special purpose combinatorial algorithms.

The *volume algorithm* presented in Barahona and Anbil (2000) is a promising new algorithm also based on Lagrangean-type relaxation. It was successfully integrated in a branch-and-cut framework to solve some difficult instances of combinatorial optimization problems (Barahona and Ladanyi, 2001).

Benders' decomposition is very often implicitly used within cutting plane algorithms, see for instance the derivation of lift-and-project cuts in Section 3.1. Other applications areas are problems whose constraint matrix has bordered block diagonal form, where we have coupling variable instead of coupling constraints, see Fig. 3, i.e., the structure of the constraint matrix is the transposed of the structure of the constraint matrix in Fig. 1. Such problems appear, for instance, in stochastic integer programming (Sherali and Fraticelli, 2002). Benders' decomposition is attractive in this case, because Benders' subproblem decomposes into k independent problems.

5 Final remarks

In this chapter we have described the state-of-the-art in solving general mixed integer programs where we put our emphasis on the branch-and-cut method.

In Section 2 we explained in detail preprocessing techniques and some ideas used in structure analysis. These are however just two steps, though important, in answering the question on how information that is inhered in a problem can be carried over to the MIP solver. The difficulty is that the only "language" that MIP solvers understand and in which information can be transmitted are linear inequalities: The MIP solver gets as input some formulation as in (1). But such a formulation might be worse than others as we have seen for the Steiner tree problem in Section 2 and there is basically no way to reformulate (3) into (4) if no additional information like "this is a Steiner tree problem" is given. In other words, there are further tools necessary in order to transmit such information. Modeling languages like `AMPL`

(Fourer, Gay, and Kernighan, 1993) or ZIMPL (Koch, 2001) are going in this direction, but more needs to be done.

In Section 3 we described several relaxation methods where we mainly concentrated on cutting planes. Although the cutting plane method is among the most successful to solve general mixed integer programs, it is not the only one and there is pressure of competition from various sides like semi-definite programming, Gomory's group approach, basis reduction or primal approaches, see the various chapters in this handbook. We explained the most frequently used cutting planes within general MIP solvers, Gomory cuts, mixed integer rounding cuts, lift-and-project cuts as well as knapsack and set packing cutting planes. Of course, there are more and the interested reader will find a comprehensive survey in Marchand et al. (2002).

Finally, we discussed the basic strategies used in enumerating the branch-and-bound tree. We have seen that they have a big influence on the performance. A bit disappointing from a mathematical point of view is that these strategies are only evaluated computationally and that there is no theoretical proof that tells that one strategy is better than another.

All in all, mixed integer programming solvers have become much better during the last years. Their success lies in the fact that they gather more and more knowledge from the solution of special purpose problems and incorporate it into their codes. This process will and must continue to push the frontier of solvability further and further.

5.1 Software

The whole chapter was about the features of current mixed integer programming solvers. So we do not want to conclude without mentioning some of them. Due to the rich variety of applications and problems that can be modeled as mixed integer programs, it is not in the least surprising that many codes exist and not just a few of them are business oriented. In many cases, free trial versions of the software products mentioned below are available for testing. From time to time, the INFORMS newsletter OR/MS Today gives a survey on currently available commercial linear and integer programming solvers, see for instance Sharda (1995).

The following list shows software where we know that it has included many of the aspects that are mentioned in this chapter:

ABACUS, developed at the University of Cologne (Thienel, 1995), provides a branch-and-cut framework mainly for combinatorial optimization problems,

bc-opt, developed at CORE (Cordier et al., 1999), is very strong for mixed 0–1 problems,

CPLEX, developed at Incline Village (Bixby et al., 1998; ILOG CPLEX Division, 2000), is one of the currently best commercial codes,

`LINDO` and `LINGO` are commercial codes developed at Lindo Systems Inc. (1997) used in many real-world applications,

`MINTO`, developed at Georgia Institute of Technology (Nemhauser, Savelsbergh, and Sigismondi, 1994), is excellent in cutting planes and has included basically all the mentioned cutting planes and more,

`MIPO`, developed at Columbia University (Balas et al., 1996), is very good in lift-and-project cuts,

`OSL`, developed at IBM Corporation (Wilson, 1992), is now available with COIN, an open source Computational Infrastructure for Operations Research (COIN, 2002),

`SIP`, developed at Darmstadt University of Technology and ZIB, is the software of one of the authors,

`SYMPHONE`, developed at Cornell University and Lehigh University (Ralphs, 2000), has its main focus on providing a parallel framework,

`XPRESS-MP`, developed at DASH (DASH Optimization, 2001), is also one of the best commercial codes.

References

Aardal, K., Y. Pochet, L. A. Wolsey (1995). Capacitated facility location: valid inequalities and facets. *Mathematics of Operations Research* 20, 562–582.

Aardal, K., R. Weismantel, L. A. Wolsey (2002). Non-standard approaches to integer programming. *Discrete Applied Mathematics* 123/124, 5–74.

Achterberg, T., T. Koch, A. Martin (2005). Branching Rules Revisited, *Operation Research Letters* 33, 42–54.

Andersen, E. D., K. D. Andersen (1995). Presolving in linear programming. *Mathematical Programming* 71, 221–245.

Applegate, D., R. E. Bixby, V. Chvátal, W. Cook (March, 1995). Finding cuts in the TSP. Technical Report 95-05, DIMACS.

Atamtürk, A. (2003). On the facets of the mixed-integer knapsack polyhedron. *Mathematical Programming* 98, 145–175.

Atamtürk, A. (2004). Sequence independent lifting for mixed integer programming. *Operations Research* 52, 487–490.

Atamtürk, A. (2002). On capacitated network design cut-set polyhedral. *Mathematical Programming* 92, 425–437.

Atamtürk, A., G. L. Nemhauser, M. W. P. Savelsbergh (2000). Conflict graphs in integer programming. *European Journal of Operations Research* 121, 40–55.

Balas, E. (1975). Facets of the knapsack polytope. *Mathematical Programming* 8, 146–164.

Balas, E., S. Ceria, G. Cornuejols (1993). A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical Programming* 58, 295–324.

Balas, E., S. Ceria, G. Cornuejols (1996). Mixed 0–1 programming by lift-and-project in a branch-and-cut framework. *Management Science* 42, 1229–1246.

Balas, E., S. Ceria, G. Cornuejols, N. Natraj (1996). Gomory cuts revisited. *Operations Research Letters* 19, 1–9.

Balas, E., S. Ceria, M. Dawande, F. Margot, G. Pataki (2001). OCTANE: a new heuristic for pure 0–1 programs. *Operations Research* 49, 207–225.

Balas, E., R. Martin (1980). Pivot and complement: a heuristic for 0–1 programming. *Management Science* 26, 86–96.

Balas, E., E. Zemel (1978). Facets of the knapsack polytope from minimal covers. *SIAM Journal on Applied Mathematics* 34, 119–148.

Barahona, F., L. Ladanyi (2001). Branch and cut based on the volume algorithm: Steiner trees in graphs and max-cut. Technical Report RC22221, IBM.

Barahona, F., Ranga Anbil (2000). The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming* 87(3), 385–399.

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

Benders, J. F. (1962). Partitioning procedures for solving mixed variables programming. *Numerische Mathematik* 4, 238–252.

Benichou, M., J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, O. Vincent (1971). Experiments in mixed-integer programming. *Mathematical Programming* 1, 76–94.

Bienstock, D., M. Zuckerberg (2003). Subset algebra lift operators for 0–1 integer programming. Technical Report CORC Report 2002-01, Columbia University, New York.

Bixby, R. E. (1994). *Lectures on Linear Programming*. Rice University, Houston, Texas, Spring.

Bixby, R. E., S. Ceria, C. McZeal, M. W. P. Savelsbergh (1998). An updated mixed integer programming library: MIPLIB 3.0. Paper and Problems are available at WWW Page: http://www.caam.rice.edu/∼bixby/miplib/miplib.html.

Bixby, R. E., M. Fenelon, Z. Guand, E. Rothberg, R. Wunderling (1999). MIP: theory and practice closing the gap. Technical Report, ILOG Inc., Paris, France.

Borndörfer, R. (1998). *Aspects of Set Packing, Partitioning, and Covering*. Shaker, Aachen.

Borndörfer, R., C. E. Ferreira, A. Martin (1998). Decomposing matrices into blocks. *SIAM Journal on Optimization* 9, 236–269.

Ceria, S., C. Cordier, H. Marchand, L. A. Wolsey (1998). Cutting planes for integer programs with general integer variables. *Mathematical Programming* 81, 201–214.

Chopra, S., M. R. Rao (1994). The Steiner tree problem I: formulations, compositions and extension of facets. *Mathematical Programming* 64(2), 209–229.

Clochard, J. M., D. Naddef (1993). Using path inequalities in a branch-and-cut code for the symmetric traveling salesman problem, in: L. A. Wolsey, G. Rinaldi (eds.), *Proceedings on the Third IPCO Conference* 291–311.

COIN (2002). A COmputational INfrastructures for Operations Research. URL: http://www124.ibm.com/developerworks/opensource/coin.

Cordier, C., H. Marchand, R. Laundy, L. A. Wolsey (1999). bc – opt: a branch-and-cut code for mixed integer programs. *Mathematical Programming* 86, 335–354.

Crowder, H., E. Johnson, M. W. Padberg (1983). Solving large-scale zero-one linear programming problems. *Operations Reserch* 31, 803–834.

Dantzig, G. B., P. Wolfe (1960). Decomposition principle for linear programs. *Operations Research* 8, 101–111.

DASH Optimization (2001). Blisworth House, Church Lane, Blisworth, Northants NN7 3BX, UK. *XPRESS-MP Optimisation Subroutine Library*, Information available at URL http://www.dash.co.uk.

de Farias, I. R., E. L. Johnson, G. L. Nemhauser (2002). Facets of the complementarity knapsack polytope. *Mathematics of Operations Research*, 27, 210–226.

Eckstein, J. (1994). Parallel branch-and-bound algorithms for general mixed integer programming on the CM-5. *SIAM Journal on Optimization* 4, 794–814.

Ferreira, C. E. (1994). *On Combinatorial Optimization Problems Arising in Computer System Design*. PhD thesis, Technische Universität, Berlin.

Ferreira, C. E., A. Martin, R. Weismantel (1996). Solving multiple knapsack problems by cutting planes. *SIAM Journal on Optimization* 6, 858–877.

Fischetti, M., A. Lodi (2002). Local branching. *Mathematical Programming* 98, 23–47.
Fourer, R., D. M. Gay, B. W. Kernighan (1993). *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press/Brooks/Cole Publishing Company.
Fulkerson, D. R. (1971). Blocking and anti-blocking pairs of polyhedra. *Mathematical Programming* 1, 168–194.
Garey, M. R., D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.
Goffin, J. L., J. P. Vial (1999). Convex nondifferentiable optimization: a survey focused on the analytic center cutting plane method. Technical Report 99.02, Logilab, Universite de Geneve. To appear in *Optimization Methods and Software*.
Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society* 64, 275–278.
Gomory, R. E. (1960). An algorithm for the mixed integer problem. Technical Report RM-2597, The RAND cooperation.
Gomory, R. E. (1960). Solving linear programming problems in integers, in: R. Bellman M. Hall (eds.), *Combinatorial Analysis, Proceedings of Symposia in Applied Mathematics* Vol. 10, Providence RI.
Gondzio, J. (1997). Presolve analysis of linear programs prior to apply an interior point method. *INFORMS Journal on Computing* 9, 73–91.
Grötschel, M., L. Lovaśz, A. Schrijver (1988). *Geometric Algorithms and Combinatorial Optimization*. Springer.
Grötschel, M., C. L. Monma, M. Stoer (1992). Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations Research* 40, 309–330.
Gu, Z., G. L. Nemhauser, M. W. P. Savelsbergh (1998). Cover inequalities for 0–1 linear programs: complexity. *INFORMS Journal on Computing* 11, 117–123.
Gu, Z., G. L. Nemhauser, M. W. P. Savelsbergh (1998). Cover inequalities for 0–1 linear programs: computation. *INFORMS Journal on Computing* 10, 427–437.
Gu, Z., G. L. Nemhauser, M. W. P. Savelsbergh (1999). Lifted flow cover inequalities for mixed 0–1 integer programs. *Mathematical Programming* 85, 439–468.
Gu, Z., G. L. Nemhauser, M. W. P. Savelsbergh (2000). Sequence independent lifting in mixed integer programming. *Journal on Combinatorial Optimization* 4, 109–129.
Hammer, P. L., E. Johnson, U. N. Peled (1975). Facets of regular 0–1 polytopes. *Mathematical Programming* 8, 179–206.
Held, M., R. Karp (1971). The traveling-salesman problem and minimum spanning trees: part II. *Mathematical Programming* 1, 6–25.
Hiriart-Urruty, J. B., C. Lemarechal. (1993). Convex analysis and minimization algorithms, part 2: advanced theory and bundle methods. *Grundlehren der Mathematischen Wissenschaften*. Springer-Verlag, Vol. 306.
Hoffman, K. L., M. W. Padberg (1991). Improved LP-representations of zero-one linear programs for branch-and-cut. *ORSA Journal on Computing* 3, 121–134.
ILOG CPLEX Division (1997). 889 Alder Avenue, Suite 200, Incline Village, NV 89451, USA. *Using the CPLEX Callabel Library*, Information available at URL http://www.cplex.com.
ILOG CPLEX Division (2000). 889 Alder Avenue, Suite 200, Incline Village, NV 89451, USA. *Using the CPLEX Callabel Library*, Information available at URL http://www.cplex.com.
Johnson, E., M. W. Padberg (1981). A note on the knapsack problem with special ordered sets. *Operations Research Letters* 1, 18–22.
Johnson, E. L., G. L. Nemhauser, M. W. P. Savelsbergh (2000). Progress in linear programming based branch-and-bound algorithms: an exposition. *INFORMS Journal on Computing* 12, 2–23.
Klabjan, D., G. L. Nemhauser, C. Tovey (1998). The complexity of cover inequality separation. *Operations Research Letters* 23, 35–40.
Koch, T. (2001). ZIMPL user guide. Technical Report Preprint 01-20, Konrad-Zuse-Zentrum Für Informationstechnik Berlin.

Koch, T., A. Martin, S. Voß (2001). SteinLib: an updated library on Steiner tree problems in graphs, in: D.-Z. Du, X. Cheng (eds.), *Steiner Tress in Industries*, Kluwer, 285–325.

Land, A., S. Powell (1979). Computer codes for problems of integer programming. *Annals of Discrete Mathematics* 5, 221–269.

Lasserre, J. B. (2001). An explicit exact SDP relaxation for nonlinear 0–1 programs, in: K. Aardal, A. M. H. Gerards (eds.), *Lecture Notes in Computer Science*, 293–303.

Lemaréchal, C., A. Renaud (2001). A geometric study of duality gaps, with applications. *Mathematical Programming* 90, 399–427.

Linderoth, J. T., M. W. P. Savelsbergh (1999). A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing* 11, 173–187.

Lindo Systems Inc. (1997). *Optimization Modeling with LINDO*. See web page: http://www.lindo.com.

Löbel, A. (1997). *Optimal Vehicle Scheduling in Public Transit*. PhD thesis, Technische Universität Berlin.

Löbel, A. (1998). Vehicle scheduling in public transit and lagrangean pricing. *Management Science* 12(44), 1637–1649.

Lovász, L., A. Schrijver (1991). Cones of matrices and set-functions and 0–1 optimization. *SIAM Journal on Optimization* 1, 166–190.

Lübbecke, J. E., Jacques Desrosiers (2002). Selected topics in column generation. Technical Report, Braunschweig University of Technology, Department of Mathematical Optimization.

Marchand, H. (1998). *A Polyhedral 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., A. Martin, R. Weismantel, L. A. Wolsey (2002). Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics* 123/124, 391–440.

Marchand, H., L. A. Wolsey (1999). The 0–1 knapsack problem with a single continuous variable. *Mathematical Programming* 85, 15–33.

Marchand, H., L. A. Wolsey (2001). Aggregation and mixed integer rounding to solve MIPs. *Operations Research* 49, 363–371.

Martin, A. (1998). Integer programs with block structure. Habilitations-Schrift, Technische Universität Berlin, Available as ZIB-Preprint SC-99-03, see www.zib.de.

Martin, A., R. Weismantel (1998). The intersection of knapsack polyhedra and extensions, in: R. E., Bixby, E. A. Boyd, R. Z. Fíos-Mercado (eds.), *Integer Programming and Combinatorial Optimization*, Proceedings of the 6th IPCO Conference, 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.

Naddef, D. (2002). Polyhedral theory and branch-and-cut algorithms for the symmetric tsp. in: G. Gutin, A. Punnen (eds.), *The Traveling Salesman Problem and its Variations*. Kluwer.

Nemhauser, G. L., M. W. P. Savelsbergh, G. Minto, C. Sigismondi (1994). MINTO a mixed integer optimizer. *Operations Research Letters* 15, 47–58.

Nemhauser, G. L., P. H. Vance (1994). Lifted cover facets of the 0–1 knapsack Polytope with GUB constraints. *Operations Research Letters* 16, 255–263.

Nemhauser, G. L., L. A. Wolsey (1988). *Integer and Combinatorial Optimization*. Wiley.

Nemhauser, G. L., L. A. Wolsey (1990). A recursive procedure to generate all cuts for 0–1 mixed integer programs. *Mathematical Programming* 46, 379–390.

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. *OR* 23(4), 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. (2001). Classical cuts for mixed-integer programming and branch-and-cut. *Mathematical Methods of OR* 53, 173–203.

Pedberg, M. W., T. J. Van Roy, L. A. Wolsey (1985). Valid inequalities for fixed charge problems. *Operations Research* 33, 842–861.

Pochet, Y. (1988). Valid inequalities and separation for capacitated economic lot-sizing. *Operations Research Letters* 7, 109–116.

Ralphs, T. K. (September, 2000). *SYMPHONY Version* 2.8 *User's Manual*. Information available at http://www.lehigh.edu/inime/ralphs.htm.

Richard, J. P., I. R. de Farias, G. L. Nemhauser (2001). Lifted inequalities for 0–1 mixed integer programming: basic theory and algorithms. *Lecture Notes in Computer Science*.

Van Roy, T. J., L. A. Wolsey (1986). Valid inequalities for mixed 0–1 programs. *Discrete Applied Mathematics* 4, 199–213.

Van Roy, T. J., L. A. Wolsey (1987). Solving mixed integer programming problems using automatic reformulation. *Operations Research* 35, 45–57.

Vasek Chvátal (1983). *Linear Programming*. W. H. Freeman and Company.

Savelsbergh, M. W. P. (1994). Preprocessing and probing for mixed integer programming problems. *ORSA J. on Computing* 6, 445–454.

Savelsbergh, M. W. P. (2001). Branch-and-price: integer programming with column generation, in: P. Pardalos, C. Flouda (eds.), *Encylopedia of Optimization,* Kluwer.

Schrijver, A. (1986). *Theory of Linear and Integer Programming*. Wiley, Chichester.

Sharda, R. (1995). Linear programming solver software for personal computers: 1995 report. *OR/MS Today* 22(5), 49–57.

Sherali, H., W. Adams (1990). A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal of Discrete Mathematics* 3, 411–430.

Sherali, H. D., B. M. P. Fraticelli (2002). A modification of Benders' decomposition algorithm for discrete subproblems: an approach for stochastic programs with integer recourse. *Journal of Global Optimization* 22, 319–342.

Suhl, U. H., R. Szymanski (1994). Supernode processing of mixed-integer models. *Computational Optimization and Applications* 3, 317–331.

Thienel, S. (1995). *ABACUS A Branch-And-Cut System*. PhD thesis, Universität zu Köln.

Vanderbeck, F. (1999). Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming* 46, 565–594.

Vanderbeck, F. (2000). On Dantzig–Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research* 48(1), 111–128.

Weismantel, R. (1997). On the 0/1 knapsack polytope. *Mathematical Programming* 77(1), 49–68.

Wilson, D. G. (1992). A brief introduction to the ibm optimization subroutine library. *SIAG/OPT Views and News* 1, 9–10.

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.

Zemel, E. (1989). Easily computable facets of the knapsack polytope. *Mathematics of Operations Research* 14, 760–764.

Zhao, X., P. B. Luh (2002). New bundle methods for solving Lagrangian relaxation dual problems. *Journal of Optimization Theory and Applications* 113(2), 373–397.