Computational Integer Programming and Cutting Planes

Armin Fügenschuh   Alexander Martin

September 19, 2001

1 Introduction

The study and solution of linear integer programs lies in the heart of discrete optimization. Various problems in science, technology, business, and society can be modeled as linear integer programming problems and their number is tremendous and still increasing. It is not in the least surprising that there is no unique method that solves all integer programming problems. This handbook, for instance, documents the variety of ideas, approaches and methods that help to solve integer programs, see also the surveys [2, 55]. Among the most successful methods are currently LP based branch-and-bound algorithms where the underlying linear programs (LPs) are possibly strengthened by cutting planes. For example, most commercial integer programming solvers, see [76], or special purpose codes for problems like the traveling salesman problem are based on this method.
The purpose of this paper is to describe the main ingredients of todays (commercial or research oriented) solvers for integer programs. Consider an integer program or more general a mixed integer program (MIP) in the form

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

where M, N and C are finite sets with N and C disjoint, $A \in \mathbb{R}^{M \times (N \cup C)}$, $c, l, u \in \mathbb{R}^{N \cup C}$, $b \in \mathbb{R}^M$. An integer variable $x_j \in \mathbb{Z}$ with $l_j = 0$ and $u_j = 1$ is called *binary*. If some variable x_j has no lower or upper bound, we assume $l_j = -\infty$ or $u_j = +\infty$, respectively.

Usually, (1) models a problem arising in some application and the formulation for modeling this problem is not unique. In fact, for one and the same problem various formulations might exist and the first task we are faced with is 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 no more obviously redundant information.

The route we follow to solve an $\mathcal{NP}$-hard problem like (1) to optimality is to attack it from two sides. First, we consider the dual side and determine a lower bound on the objective function by relaxing the problem. The basic idea of all relaxation methods is to get rid of the part of the problem that makes it difficult. The methods differ in which part to delete and in the way to reintroduce the deleted part. The most common 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 part 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 integer 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 branch-and-bound. 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 described outline is not the only way for the solution of (1), but it is definitely the most used and very often among the most successful. Other ideas 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 the "right" formulation. The right formulation is of course not unique and it strongly depends on the solution method one wants to use to solve the problem. The method we mainly focus on in this paper 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, see (10) for a definition. There are no general rules like the fewer the number of variables and/or constraints the better the formulation. Very often a theoretical analysis is necessary to get the right insides. In the following we exemplarily discuss a classical problem from combinatorial optimization, the Steiner tree problem, which underpins the statement that fewer variables is 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 from s to 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. The *Steiner tree problem* is to find a minimal Steiner tree with respect to some given edge costs $c_e, e \in E$. 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

$$\begin{array}{lll}
\min & c^T x & \\
& x(\delta(W)) \geq 1, & \text{for all } W \subset V, W \cap T \neq \emptyset, \\
& & (V \setminus W) \cap T \neq \emptyset, \\
& 0 \leq x_e \leq 1, & \text{for all } e \in E, \\
& x \text{ integer},
\end{array} \tag{2}$$

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 in 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, the Steiner tree problem can be solved via (2). 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 anti-parallel 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 arborescences in D which contain at most one of two anti-parallel arcs. 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

$$\begin{array}{lll} \min & \vec{c}^T y & \\ & y(\delta^+(W)) \geq 1, & \text{for all } W \subset V, r \in W, \\ & & (V \setminus W) \cap T \neq \emptyset, \\ & 0 \leq y_a \leq 1, & \text{for all } a \in A, \\ & y \text{ integer}, & \end{array} \tag{3}$$

where $\delta^+(X) := \{(u,v) \in A \mid u \in X, v \in V \setminus X\}$ 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 (3). Which of the two models (2) and (3) should be used to solve the Steiner tree problem in graphs?
At first glance, (2) is preferable to (3), since it contains only half the number of variables and basically the same number of constraints. However, it turns out that the optimal value of the LP relaxation of the directed model $z_d := \min\{\vec{c}^T y \mid y$ satisfies the trivial and directed Steiner cut inequalities$\}$ is greater than or equal to the corresponding value of the undirected formulation $z_u := \min\{c^T x \mid x$ satisfies the trivial and undirected Steiner cut inequalities$\}$. Even, if the undirected formulation is tightened by the so-called Steiner partition inequalities, this relation holds [18]. This is even more astonishing, since the separation problem of the Steiner partition inequalities is difficult ($\mathcal{NP}$-hard), see [36], whereas the directed Steiner cut inequalities can be separated in polynomial time by max flow computation. Finally, the disadvantage of the directed model that the number of variables is doubled is not really a bottleneck. Since we are minimizing a non-negative objective function, the variable of one of two anti-parallel arcs will usually be at its lower bound and rarely touched by the solution algorithm. Thus, the directed model is much better than the undirected, though it contains more variables. And in fact, most state-of-the-art solvers for the Steiner tree problem in graphs use formulation (3) or one that is equivalent to (3), see [49] for further references.
The Steiner tree problem shows that it is not easy to find a tight problem formulation and that often a non-trivial analysis is necessary to come to a good decision.

Once we have decided on some formulation we face the next step, the step 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 non-trivial example 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 [3, 12, 22, 43, 78]. We denote by $s_i \in \{\leq, =\}$ the sign of row i, i. e., (1) reads $\min\{c^T x : Ax\, s\, b,\ l \leq x \leq u,\ x \in \mathbb{Z}^N \times \mathbb{R}^C\}$. 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 = '$\leq$', and $a_{ij} = 0$ if s_i = '$=$' for $i \in M$. If $l_j > -\infty$, we can fix column j to its lower bound. If $l_j = -\infty$ the problem is unbounded or infeasible. The same arguments apply to some column j with $c_j \leq 0$. Suppose $a_{ij} \leq 0$ if s_i = '$\leq$', $a_{ij} = 0$ if s_i = '$=$' for $i \in M$. If $u_j < \infty$, we can fix column j to its upper bound. If $u_j = \infty$ the problem is unbounded or infeasible.

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

$$
\begin{aligned}
L_i &= \sum_{j \in P_i} a_{ij} l_j + \sum_{j \in N_i} a_{ij} u_j \\
U_i &= \sum_{j \in P_i} a_{ij} u_j + \sum_{j \in N_i} a_{ij} l_j
\end{aligned}
\tag{4}
$$

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

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

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

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

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

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

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

Let $\bar{u}_j = \min_i \bar{u}_{ij}$ and $\bar{l}_j = \max_i \bar{l}_{ij}$. If $\bar{u}_j \leq u_j$ and $\bar{l}_j \geq l_j$, we speak of an *implied free variable*. The simplex method might benefit from not updating the bounds but treating variable x_j as a free variable (note, setting the bounds of j to $-\infty$ and $+\infty$ will not change the feasible region). Free variables will always be in the basis and are thus useful in finding a starting basis. For mixed integer programs however, it is 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.

Coefficient Reduction. The row bounds in (4) can also be used to reduce coefficients of binary variables. Consider some row i with $s_i = \text{`}\leq\text{'}$ and let x_j be a binary variable with $a_{ij} \neq 0$.

$$\text{If} \begin{cases} a_{ij} < 0, U_i + a_{ij} < b_i, & \text{set} \quad a'_{ij} = b_i - U_i, \\ a_{ij} > 0, U_i - a_{ij} < b_i, & \text{set} \quad \begin{cases} a'_{ij} = U_i - b_i, \\ b_i = U_i - a_{ij}, \end{cases} \end{cases} \tag{5}$$

[1] `Miplib` is a public available test set of real-world mixed integer programming problems [13].

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

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

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

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

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

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

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

$$\frac{b_i - a_{ij}x_j}{a_{ik}} \, . \tag{6}$$

In case x_k is binary or integer, the substitution is only possible, if the term (6) is guaranteed to be binary or integer as well. If this is true or x_k is a continuous variable, we aggregate the two variables. The new bounds of variable x_j are $l_j = \max\{l_j, (b_i - a_{ik}l_k)/a_{ij}\}$ and $u_j = \min\{u_j, (b_i - a_{ik}u_k)/a_{ij}\}$ if $a_{ik}/a_{ij} < 0$, and $l_j = \max\{l_j, (b_i - a_{ik}u_k)/a_{ij}\}$ and $u_j = \min\{u_j, (b_i - a_{ik}l_k)/a_{ij}\}$ if $a_{ik}/a_{ij} > 0$.

Of course, aggregation can also be applied to equations whose support is greater than two. However, this might cause additional fill in the matrix. Hence, aggregation is usually restricted to constraints and columns with small support.

Disaggregation. Disaggregation of columns is to our knowledge 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 [34].

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)

$$x_{85} + x_{90} + x_{95} + x_{100} + x_{217} + x_{222} + x_{227} + x_{232} - 8x_{246} \leq 0 \qquad (7)$$

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 (7) by the following eight inequalities:

$$x_i - x_{246} \leq 0 \qquad \text{for all } i \in S. \qquad (8)$$

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 (8) can be generated on demand.

Probing. Probing is sometimes used in general mixed integer programming codes, see, for instance, [78]. The idea is to set some binary variable temporarily to zero or one and try to deduce further fixings from that. These implications can be expressed in inequalities as follows:

$$\begin{aligned} (x_j = 1 \Rightarrow x_i = \alpha) \quad &\Rightarrow \begin{cases} x_i \geq & l_i + (\alpha - l_i)x_j \\ x_i \leq & u_i - (u_i - \alpha)x_j \end{cases} \\ (x_j = 0 \Rightarrow x_i = \alpha) \quad &\Rightarrow \begin{cases} x_i \geq & \alpha - (\alpha - l_i)x_j \\ x_i \leq & \alpha + (u_i - \alpha)x_j \end{cases} \end{aligned} \qquad (9)$$

As an example, suppose we set in (7) variable x_{246} temporary to zero. This implies that $x_i = 0$ for all $i \in S$. Applying (9) we deduce the inequality

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

for all $i \in S$ which is exactly (8).

Besides the described cases, 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 formulation, because better formulations are straight-forward to derive. But such formulation 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 non-experts 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 [58], 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 non-zeros by around 10% on average. The time spent in preprocessing is neglectable (below one per mill). Interesting to note is also that for some problems presolve is indispensable for their solution. For example, problem fixnet6 from the `Miplib` is an instance, where most solvers fail without preprocessing, but with presolve the instance turns out to be very easy.

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. Interesting to note is 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 want to discuss one case, which shows that there might be more potential for MIPs. Consider a matrix in so-called bordered block diagonal form as depicted in Figure 1.

Suppose the constraint matrix of (1) has such a form and suppose even that there are just a few or even no coupling constraints. In the latter case the problem decomposes into number of blocks many independent problems, which can be solved much faster than the original problem. Even if there are coupling constraints this structure might help for instance to derive new cutting planes. The question arises

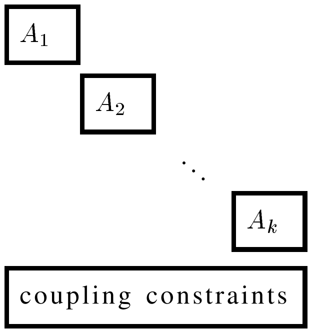

Figure 1: Matrix in bordered block diagonal form

do MIPs have such a structure, possibly after reordering columns and rows? There are, of course, some obvious cases, where the matrix is already in this form, like multi-commodity flow problems, multiple knapsack problems or other packing problems. But, there are problems where bordered block diagonal form is hidden in the problem formulation (1) and can only be detected after reordering columns and rows. In [16] Borndörfer et al. have analyzed this question and checked whether matrices from MIPs can be brought into this form. They have tested on various instances, especially on 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 developed heuristics for detecting such a form are fast enough to be incorporated into preprocessing of a MIP solver. Martin and Weismantel [58, 59] 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 this section we attack (1) 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, Lagrange relaxation as well as Dantzig-Wolfe and Benders' decomposition. The idea of them 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 $N = \{1, \dots, p\}$ and $C = \{p+1, \dots, n\}$. If we drop the integrality condition on the variables $x_1, \dots, x_p$, we obtain the so-called *linear programming relaxation*, or *LP relaxation* for short:

$$\begin{aligned} z_{\text{LP}} = \min \quad & c^T x \\ \text{s.t.} \quad & Ax \le b \\ & x \in \mathbb{R}^n . \end{aligned} \tag{10}$$

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

The crucial point in the theory of solving general mixed integer problems is a sufficient good understanding of the underlying polyhedrons. To problem (1) we associate the polyhedron $P_{\text{MIP}} := \operatorname{conv}\{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} : Ax \le b\}$, i. e., the convex hull of all feasible points for (1). In the same way we define the associated polyhedron of problem (10) by $P_{\text{LP}} := \operatorname{conv}\{x \in \mathbb{R}^n : Ax \le b\}$. Of course, $P_{\text{MIP}} \subset P_{\text{LP}}$ and $z_{\text{LP}} \le z_{\text{MIP}}$, so P_{LP} can be viewed as a relaxation of P_{MIP}. Our main goal is to tighten this approximation.

Note that if $x^* = (x_1^*, \dots, x_n^*)$ is an optimal solution of (10) 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 $x_1^*, \dots, x_p^*$ 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 found a cutting plane $a^T x \le \beta$, we add it to the problem (10) and obtain

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

which strengthens (10) in the sense that $P_{\text{LP}} \subset P_{\text{LP}^1} \subseteq P_{\text{MIP}}$, where $P_{\text{LP}^1} := \text{conv}\{x : Ax \leq b, a^T x \leq \beta\}$ 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), which we now call integer program (IP). The algorithm we present in the sequel is making use of information given by the simplex algorithm. Hereto we transform the problem into standard form by adding slack variables. As we will see below, its main step is based on integer rounding, but this step only works accurate if the constrained matrix A and the right-hand side b are integral (for computational issues, this isn't really a restriction) and all variables x have to be non-negative. Summing up, we talk about the following problem:

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

with $A \in \mathbb{Z}^{n \times m}$ and $b \in \mathbb{Z}^m$. We call the associated polyhedron $P^{\text{St}}_{\text{IP}} := \text{conv}\{x \in \mathbb{Z}^n_+ : Ax = b\}$.

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 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 non-basic variables $N = \{1, \ldots, n\} \backslash B$. If x^* is integral, we 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 for the i-th variable of x_B.

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

where $A_{i\cdot}^{-1}$ denotes the i-th row of A^{-1} and $A_{\cdot j}$ the j-th column of A, respectively. We set $\bar{b}_i := A_{i\cdot}^{-1} b$ and $\bar{a}_{ij} := A_{i\cdot}^{-1} A_{\cdot j}$ 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 was assumed to be integral 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 $P_{\text{IP}}^{\text{St}}$, but it cuts off x^*, since $x^*_i = \bar{b}_i \notin \mathbb{Z}, x^*_j = 0$ for all $j \in N$ and $\lfloor \bar{b}_i \rfloor < \bar{b}_i$. Named after its inventor, inequalities of this type are called *Gomory cuts* [31, 33]. 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 constrained matrix, the right-hand side and the new slack variable have to be integral. Gomory showed that after repeating this steps a finite number of times, an integer optimal solution is found.

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 $P_{\text{MIP}}^{\text{St}} := \text{conv}\{x \in \mathbb{Z}^p_+ \times \mathbb{R}^{n-p}_+ : Ax = b\}$, if x cannot be assumed to be integral. For the general mixed-integer case, we develop three different methods to obtain valid inequalities. They are all more or less based on the following disjunctive argument.

Lemma 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 $\operatorname{conv}(P \cup Q)$.

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 $P_{\text{MIP}}^{\text{St}}$ 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 non-basic 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

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

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

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

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 $P := P^{\text{St}}_{\text{MIP}} \cap \{x : \sum_{j\in N} \bar{a}_{ij}x_j \geq 0\}$ and $Q := P^{\text{St}}_{\text{MIP}} \cap \{x : \sum_{j\in N} \bar{a}_{ij}x_j \leq 0\}$. Because of $\max(\bar{a}_{ij}, 0) = \bar{a}_{ij}$ for $j \in N^+$ and $\max(-\frac{f(\bar{b}_i}{1-f(\bar{b}_i)}\bar{a}_{ij}, 0) = -\frac{f(\bar{b}_i)}{1-f(\bar{b}_i)}\bar{a}_{ij}$ for $j \in N^-$ we obtain the valid inequality for $P^{\text{St}}_{\text{MIP}}$

$$\sum_{j\in N^+} \bar{a}_{ij}x_j - \frac{f(\bar{b}_i)}{1-f(\bar{b}_i)} \sum_{j\in N^-} \bar{a}_{ij}x_j \geq f(\bar{b}_i), \tag{18}$$

which cuts off x^*. It is possible to strengthen inequality (18) in the following way. Observe that the derivation of it does not change, if we add integer multiples to those variables x_j, $j \in N$, that are integral (only the value of k 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 gap between right-hand and left-hand side is now as small as possible. In N^- the final coefficient is $-\frac{f(\bar{b}_i)}{1-f(\bar{b}_i)}\bar{a}_{ij}$, so the smallest gap is achieved by the factor $\frac{f(\bar{b}_i)(1-f(\bar{a}_{ij}))}{1-f(\bar{b}_i)}$. We still have the freedom to select between N^+ and N^-, but we will obtain the best possible coefficients by using $\min(f(\bar{a}_{ij}), \frac{f(\bar{b}_i)(1-f(\bar{a}_{ij}))}{1-f(\bar{b}_i)})$. Putting all this together yields Gomory's mixed integer cut [32]:

$$\begin{array}{lllll} \displaystyle\sum_{\substack{j\in N,\text{ integer:}\\ f(\bar{a}_{ij})\leq f(\bar{b}_i)}} f(\bar{a}_{ij})\, x_j & + & \displaystyle\sum_{\substack{j\in N,\text{ integer:}\\ f(\bar{a}_{ij})> f(\bar{b}_i)}} \frac{f(\bar{b}_i)(1-f(\bar{a}_{ij}))}{1-f(\bar{b}_i)}\, x_j & + & \\ \displaystyle\sum_{\substack{j\in N^+\\ j\text{ non-integer}}} \bar{a}_j x_j & - & \displaystyle\sum_{\substack{j\in N^-\\ j\text{ non-integer}}} \frac{f(\bar{b}_i)}{1-f(\bar{b}_i)}\, \bar{a}_j x_j & \geq & f(\bar{b}_i). \end{array} \tag{19}$$

Gomory 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 $x \in \{x \in \mathbb{Z}^p_+ \times \mathbb{R}^{n-p}_+ : Ax = b\}$.
Though Gomory's mixed integer cuts are known since the sixties they received their computational breakthrough in the nineties with the paper by [8]. In the meantime they are incorporated in basically every MIP solver, see, for instance [14]. Note that Gomory's mixed integer cuts can always be applied, the separation problem for the optimal LP solution is easy. However, adding these inequalities might cause numerical difficulties, see the discussion in [69].

3.1.3 Mixed-Integer-Rounding Cuts

We start developing the idea of this kind of cutting planes by considering the subset $X := \{(x, y) \in \mathbb{Z} \times \mathbb{R}_+ : x - y \leq b\}$ of $\mathbb{R}^2$ with $b \in \mathbb{R}$. We split $\operatorname{conv}(X)$ into two disjoint subsets $P := \operatorname{conv}(X \cap \{(x,y) : x \leq \lfloor b \rfloor\})$ and $Q := \operatorname{conv}(X \cap \{(x,y) : x \geq \lfloor b \rfloor + 1\})$. For P the inequalities $x - \lfloor b \rfloor \leq 0$ and $0 \leq y$ 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

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

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

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

Now the disjunctive argument, Lemma 2, implies that $(x - \lfloor b \rfloor)(1 - f(b)) \leq y$, or equivalently:

$$x - \frac{1}{1 - f(b)} y \leq \lfloor b \rfloor \tag{20}$$

is valid for $\operatorname{conv}(P \cup Q) = \operatorname{conv}(X)$.
From this basic situation we change now to more general settings. Consider the mixed integer set $X := \{(x,y) \in \mathbb{Z}^n_+ \times \mathbb{R}_+ : a^T x - y \leq b\}$ with $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. We define a partition of $\{1, \dots, n\}$ by $N^1 := \{i \in \{1, \dots, n\} : f(a_i) \leq f(b)\}$ and $N^2 := \{1, \dots, n\} \backslash N^1$. 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 $\lceil a_i \rceil - \lfloor a_i \rfloor = 1$)

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

and (20) yields

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

Substituting w and z gives

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

Easy computation shows that this is equivalent to

$$\sum_{i=1}^{n}\big(\lfloor a_i\rfloor + \frac{\max(0, f(a_i)-f(b))}{1-f(b)}\big)x_i - \frac{1}{1-f(b)}y \le \lfloor b\rfloor.$$

Thus we showed that this is a valid inequality for $\mathrm{conv}(X)$, the *mixed integer rounding (MIR) inequality*.
Nemhauser and Wolsey [65] 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 [54, 56] show the computational merits of MIP 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 problem space but in a higher dimensional (lifting). By projecting these inequalities back to the original space tighter inequalities can be obtained. Many different versions on how to lift and how to project back can be found in literature [6, 53, 77]. The method we want to review in detail is due to Balas et al. [6, 7]. It is based on the following observation:

Lemma 3 *If $\alpha + a^T x \ge 0$ and $\beta + b^T x \ge 0$ are valid for a polyhedron P, then $(\alpha + a^T x)(\beta + b^T x) \ge 0$ is also valid for P.*

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

Algorithm 4 *(Lift-and-Project)*

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

$$\begin{aligned}(Ax)x_j &\le bx_j \\ (Ax)(1-x_j) &\le b(1-x_j)\end{aligned} \tag{21}$$

3. *Lifting: substitute $x_i x_j$ by y_i for $i \in \{1, \dots, n\}\backslash\{j\}$ 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 [6] it is proven that $P_j = \text{conv}(P \cap \{x \in \mathbb{R}^n : x_j \in \{0,1\})$, i. e., the j-th component of each vertex of P_j is either zero or one. Moreover, they showed that a repeated application of Algorithm 4 on the first p variables yields

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

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 exist 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

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

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:

$$\begin{array}{ll} \max & u^T(Dx^* - d) \\ \text{s.t.} & u^T B = 0 \\ & u \in \mathbb{R}^n_+. \end{array} \tag{22}$$

The set $C := \{u \in \mathbb{R}^n_+ : u^T B = 0\}$ in which we are looking for the optimum is a polyhedral cone. So 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 (22) 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 [6, 7].

3.1.5 Knapsack Inequalities

The cutting planes discussed so far had one thing in common: they do not make use of 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 (but not always) better in the sense that they lead to better approximations of P_{MIP} than the general purpose cuts discussed before.

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:

$$P_{\mathrm{K}}(N, a, \beta) := \operatorname{conv}\{x \in \{0,1\}^N : \sum_{j \in N} a_j x_j \leq \beta\}$$

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 raise 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 $C \subset N$ 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 belongs the cover inequality

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

a valid inequality for $P_{\mathrm{K}}(N, a, \beta)$. The 'best' inequalities of this type are those belonging to minimal covers, i. e., $C \subset N$ is a cover and for every $s \in C$ we have $\sum_{j \in S \setminus \{s\}} a_j \leq \beta$. In what sense are they 'best'? It was shown that minimal cover inequalities define facets of $P_{\mathrm{K}}(C, a, \beta)$, i. e., the dimension of the face that is induced by the inequality is one less than the dimension of the polytope. Non-minimal cover only give faces, but not facets. Moreover, 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 [68] 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 we will say, (S, z) is a $(1, k)$-*configuration* and we can 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 $P_{\text{K}}(S \cup \{z\}, a, \beta)$.

Extended weight inequalities. Weismantel [80] pointed out that minimal cover and $(1, k)$-configuration inequalities both have a common source. 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

$$\sum_{i \in T} a_i x_i + \sum_{i \in N \setminus T} \max(a_i - r, 0) x_i \leq a(T). \tag{23}$$

is valid for $P_{\text{K}}(N, a, \beta)$. It is called *weight inequality with respect to* T. The name *weight inequality* reflects that the coefficients of the items in T equal their original weights and the number $r := \beta - a(T)$ 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 weights one to the items in T. Denote by S the subset of $N \setminus T$ 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 $P_{\text{K}}(N, a, \beta)$, 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 [80].

The separation of minimal cover inequalities is widely discussed in the literature. The complexity of cover separation has been investigated in [27, 47, 37], whereas algorithmic and implementation issues are treated among others in [22, 38, 43, 74, 83]. 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 integer, 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 [22, 46, 82, 63, 38].

From pure integer knapsack problems we switch now to mixed $0-1$ knapsack, 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

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

with non-negative coefficients, i. e., $a_j \geq 0$ for $j \in N$ and $\beta \geq 0$.
Now let $C \subset N$ be a cover and $\lambda := \sum_{j\in C} a_j - b > 0$. Marchand and Wolsey [57] recently showed that the inequality

$$\sum_{j\in C} \min(a_j,\lambda)x_j - s \leq \sum_{j\in C} \min(a_j,\lambda) - \lambda \tag{24}$$

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 also discussed in [54, 57].
Cover inequalities appear also in other contexts. In [17] 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 [25].

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

$$X = \{(x, y) \in \{0,1\}^N \times \mathbb{R}_+^N : \sum_{j \in N} y_j \leq b,\ y_j \leq a_j x_j,\ \forall j \in N\}. \tag{25}$$

We want to demonstrate how to use the mixed knapsack inequality (24) to derive new inequalities for the polyhedron $\text{conv}(X)$. 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 (24), 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,

$$\sum_{j \in C} \big(y_j + \max(a_j - \lambda, 0)(1 - x_j)\big) \leq b. \tag{26}$$

It was shown by Padberg et al. [70] that this last inequality, called *flow cover inequality*, defines a facet of $\text{conv}(X)$, if $\max_{j \in C} a_j > \lambda$.
Flow models have been extensively studied in the literature. Various generalizations of the flow cover inequality (26) have been derived for more complex flow models. In [73], 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 [1, 71]. Flow cover inequalities have been used successfully in general purpose branch-and-cut algorithms to tighten formulations of mixed integer sets [40, 39, 74].

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 $N \subset \{1,\dots,n\}$ such that $\cup_{j\in N} X_j \subseteq X$ with $X_i \cap X_j = \emptyset$ 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 $X_j, j \in \{1,\dots,n\}$, where $a_{ij} = 1$ if and only if $i \in X_j$. Let $A := (a_{ij}) \in \{0,1\}^{m\times n}$. For the decision which subset we put into the selection N we introduce $x \in \{0,1\}^n$, with $x_j = 1$ if and only if $j \in N$. With this definitions we can state the set packing problem as the following $0-1$ integer program:

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.} & Ax \leq \mathbb{1} \\ & x \in \{0,1\}^n. \end{array} \tag{27}$$

This problem is important not only from a theoretical but also from a computational point of view: set packing problems often occur as subproblems in (mixed) integer problems. A good understanding of $0-1$ integer programs with $0-1$ matrices can substantially speed up the solution process of general mixed integer problems.

In the sequel we study the *set packing polytope* $P(A) := \operatorname{conv}\{x \in \{0,1\}^n : Ax \leq \mathbb{1}\}$ associated to (27). An interpretation of this problem in a graph theoretic sense is helpful to obtain new valid inequalities that strengthen the LP relaxation of (27). 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 ij between two nodes i and j if and only if their corresponding columns in A have a common non-zero 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 for all $i,j \in S$ holds $ij \notin E$. 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 $P(G)$.

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

(i) $P(G)$ is full dimensional.

(ii) $P(G)$ is down 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 non-negative constraints $x_j \geq 0$ induce facets of $P(G)$.

It is a well-known fact that $P(G)$ is completely described by the non-negative constraints (iii) and the edge-inequalities $x_i + x_j \leq 1$ for $ij \in E$ 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 $V_C \subset V$ 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., a cycle without chords [66]. 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 [35] for details.
A *clique* (C, E_C) in a graph $G = (V, E)$ is a subset of nodes and edges such that for every two $i, j \in C, i \neq j$ there exists an edge $ij \in E_C$. A clique (C, E_C) is said to be *maximal* if every $i \in V$ with $ij \in E$ for all $j \in C$ is already contained in 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 [30, 66]. 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 [35]. 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 [35]. Beside odd cycle, clique and OR-inequalities there are many other inequalities known for the stable set polytope. Among these are blossom, odd antihole, wheel, antiweb and web, wedge inequalities and many more. [15] 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, where some inequality that is only valid under certain variable fixings is made globally valid by applying lifting, see Section 4. Assume for simplicity that all integer variables are $0 - 1$. Consider an arbitrary polytope $P \subseteq \mathbb{R}^N$ and let $L \subset N$. Suppose we have an inequality

$$\sum_{j \in L} w_j x_j \leq w_0, \tag{28}$$

which is valid for $P_L := \text{conv}(P \cap \{x : x_j = 0 \; \forall j \in N \backslash L\})$. (Note that without loss of generality, we investigate the lifting of a variable x_j that has been set to 0, because setting x_j to 1 is equivalent to setting variable $\overline{x}_j = 1 - x_j$ to 0.) The *lifting problem* is to find lifting coefficients w_j for $j \in N \backslash L$ such that

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

is valid for P. Ideally we would like inequality (29) to be "strong", i. e., if inequality (28) defines a face of high dimension of P_L, we would like the inequality (29) 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 a sequence of $N \setminus L$ according to which we obtain compute coefficients. Let $k \in N \setminus L$ be the first index in this sequence. The coefficient w_k is computed for a given $k \in N \setminus L$ so that

$$w_k x_k + \sum_{j \in L} w_j x_j \leq w_0 \tag{30}$$

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

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

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

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 (30) is valid for $P_{L \cup \{k\}}$ if $w_k \leq \Phi_L(a_k)$, see [67, 81]. Moreover, if $w_k = \Phi_L(a_k)$ and (28) defines a face of dimension t of P_L, then (30) defines a face of $P_{L \cup \{k\}}$ of dimension $t+1$.

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

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

so in general, function Φ_L can decrease as more variables are lifted in (note that $w_j \ge 0$ for all $j \in N$ in the knapsack problem). 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 [9, 5, 41, 81, 67] and embedded in a branch-and-bound framework to solve particular types of $0-1$ integer programs to optimality [22].

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

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

Note that with $(x_k, y_k) = (0,0)$, this reduces to the flow set, see (25)

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

Now suppose that the inequality

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

is valid and facet-defining for $\operatorname{conv}(X)$.

As before, let

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

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 $\operatorname{conv}(X')$ if and only if $w_k + v_k u \leq \Psi_L(u)$ for all $0 \leq u \leq a_k$, ensuring that all the feasible points with $(x_k, y_k) = (1, u)$ satisfy the inequality.
So 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. In [39] it is also shown 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 [66]. Other uses of sequential lifting can be found in [17] 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 [58, 59] lifting is used to define (lifted) feasible set inequalities for an integer set defined by multiple integer knapsack constraints.

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 Lagrange 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 $A = \binom{A_1}{A_2}$ and $b = \binom{b_1}{b_2}$, where $A_1 \in \mathbb{R}^{m_1 \times n}, A_2 \in \mathbb{R}^{m_2 \times n}, b_1 \in \mathbb{R}^{m_1}, b_2 \in \mathbb{R}^{m_2}$ with $m_1 + m_2 = m$. Then, assuming w. l. o. g. no equality constraints (1) reads

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & A_1 x \le b_1 \\ & A_2 x \le b_2 \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}. \end{aligned} \tag{31}$$

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

$$\begin{aligned} L(\lambda) = \quad \min \quad & c^T x - \lambda^T (b_1 - A_1 x) \\ \text{s.t.} \quad & x \in P^2, \end{aligned} \tag{32}$$

where $P^2 = \{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \,:\, A_2 x \le b_2\}$. (32) is called the *Lagrangean function*. Obviously, (32) is a lower bound for (31), since for any feasible solution $\bar{x}$ of (31) we have

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

Since this holds for each $\lambda \ge 0$ we get that

$$\max_{\lambda \ge 0} L(\lambda) \tag{33}$$

yields a lower bound for (1). (33) is called *Lagrangean relaxation*. Let λ^* be an optimal solution to (33). The questions remain, how good is $L(\lambda^*)$ and how to compute λ^*. An answer to the first question gives the following equation:

$$L(\lambda^*) = \min\{c^T x \,:\, A_1 x \le b_1, x \in \operatorname{conv}(P^2)\}. \tag{34}$$

A proof of this result can be found for instance in [64, 75]. Since

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

we conclude form (34)

$$z_{\text{LP}} \le L(\lambda^*) \le z_{\text{MIP}}. \tag{35}$$

Furthermore, $z_{\text{LP}} = L(\lambda^*)$ for all objective functions c if and only if $\{x \in \mathbb{R}^n \,:\, A_2 x \le b_2\}$ is integral.

It remains to discuss how to compute $L(\lambda^*)$. From a theoretical point of view it can be shown using the equivalence of separation and optimization that $L(\lambda^*)$ can be determined in polynomial time, if $\min\{\tilde{c}^T x : x \in \operatorname{conv}(P^2)\}$ can be computed in polynomial time for any objective function $\tilde{c}$, see for instance [75]. 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 (32). Then, $g^0 = A_1 x - b_1$ is a *subgradient* for L in 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 $(g^0)^T(\lambda - \lambda^0) \geq L(\lambda^*) - L(\lambda^0) \geq 0$. This suggests in order to find λ^* to start with some λ^0, compute

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

and determine iteratively, $\lambda^0, \lambda^1, \lambda^2, \ldots$ by setting $\lambda^{k+1} = \lambda^k + \mu^k g^k$, where 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 in [64].

Of course, the quality of the Lagrangean relaxation strongly depends on the set of constraints that is relaxed. On one side, we must compute (32) for various values of λ and thus it is necessary to determine one value of $L(\lambda)$ 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 [51]. Therefore, one always must find a compromise between these two conflicting goals.

Lagrangean relaxation is also very often used if the underlying linear programs of (1) are just too big to be solved directly and even the relaxed problems in (32) are still large. Often the relaxation can be done in a way that the evaluation of (32) 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_{ij} for $ij \in E$ denote the traveling time between city

i and 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 $x_{ij} \in \{0,1\}$ equals 1 if city j is visited right after city i is left, and equals 0 otherwise, so $x \in \{0,1\}^E$. The equations

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

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

$$\sum_{\{ij \in E: i,j \in U\}} x_{ij} \leq |U| - 1$$

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 (32) combinatorial algorithms are at hand and no general LP or IP solution techniques must be used. Held and Karp [42] 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 Figure 1. If we relax the coupling constraints within a Lagrangean relaxation, the remaining matrix decomposes into k independent blocks. Thus, one value $L(\lambda)$ is the sum of k individual terms that can be determined separately. Often each single block A_i models a network flow problem, a knapsack problem or the like and can thus be solved using special purpose combinatorial algorithms.

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 *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 [23] and consider again (31), 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 $v_1, \dots, v_k$ and $e_1, \dots, e_l$ such that $P^2 = \text{conv}(\{v_1, \dots, v_k\}) + \text{cone}(\{e_1, \dots, e_l\})$. In other words, $x \in P^2$ can be written in the form

$$x = \sum_{i=1}^{k} \lambda_i v_i + \sum_{j=1}^{l} \mu_j e_j \tag{36}$$

with $\lambda_1, \dots, \lambda_k \geq 0, \sum_{i=1}^{k} \lambda_i = 1$ and $\mu_1, \dots, \mu_l \geq 0$. Substituting for x from (36) we may write (31) as

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

which is equivalent to

$$\begin{aligned}
\min \quad & \sum_{i=1}^{k} (c^T v_i)\, \lambda_i + \sum_{j=1}^{l} (c^T e_j)\, \mu_j \\
\text{s.t.} \quad & \sum_{i=1}^{k} (A_1 v_i)\, \lambda_i + \sum_{j=1}^{l} (A_1 e_j)\, \mu_j \leq b_1 \\
& \sum_{i=1}^{k} \lambda_i \qquad\qquad\qquad = 1 \\
& \qquad\qquad \lambda \in \mathbb{R}^k_+, \mu \in \mathbb{R}^l_+.
\end{aligned} \tag{37}$$

(37) is called the *master problem* of (31). Comparing formulations (31) and (37) we see that we reduced the number of constraints from m to m_1, but obtain $k + l$

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 (37). However, we can use the simplex algorithm for the solution of (37). For ease of exposition abbreviate (37) by $\min\{w^T\eta : D\eta = d\}$ 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 $B \subseteq \{1,\dots,k+l\}$, $|B| = m_1+1$, with D_B non-singular and the corresponding (feasible) solution $\eta_B^* = D_B^{-1}d$ and $\eta_N^* = 0$, where $N = \{1,\dots,k+l\} \setminus B$. Observe that $D_B \in \mathbb{R}^{(m_1+1)\times(m_1+1)}$ is (much) smaller than a basis for the original system (31) and that only a fraction of the variables (m_1+1 out of $k+l$) are possibly non-zero. 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 non-negative with $\tilde{y}$ being the solution of $y^T D_B = w_B$. The non-negativity of the reduced costs can be verified via the following linear program:

$$\begin{array}{rl} \max & (c^T - \bar{y}^T A_1)x \\ \text{s.t.} & A_2 x \leq b_2 \\ & \quad x \in \mathbb{R}^n, \end{array} \tag{38}$$

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

(i) (38) 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 $v_i, i \in \{1,\dots,k\}$, 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) (38) 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, $j \in \{1,\dots,l\}$. 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.

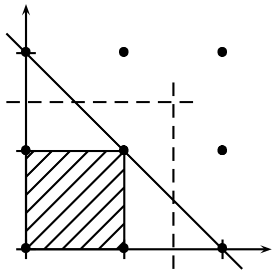

Figure 2: Extending Dantzig-Wolfe decomposition to integer programs

(iii) (38) 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 (37).

Observe that the whole problem (31) is decomposed into two problems, i. e., (37) and (38), and the approach iteratively works on the master level (37) and the subordinate level (38). The procedure starts with some feasible solution for (37) and generates new promising columns on demand by solving (38). 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 (38) turns from a linear to an integer linear program. In addition, we have to guarantee in (36) that all feasible integer solutions x of (31) can be generated by (integer) linear combinations of the vectors $v_1, \ldots, v_k$ and $e_1, \ldots, e_l$. Hereto, it is not sufficient to require λ and μ to be integer. Consider as a counterexample the problem $\max\{x_1 + x_2 : A_1 x \leq b_1, A_2 x \leq b_2, x \in \{0, 1, 2\}^2\}$ with $A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $b_1 = \binom{1.5}{1.5}$ and $A_2 = (1, 1), b_2 = 2$. In this case, $P^2 = \text{conv}(\{\binom{0}{0}, \binom{2}{0}, \binom{0}{2}\})$, see Figure 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, with the addition that all variables are $0 - 1$, this difficulty does not occur, since any $0 - 1$ solution of some $0 - 1$ polyhedron is always a vertex of that polyhedron. And in fact, column generation algorithm are not only used for the solution of large linear programs, but especially for large $0 - 1$ integer programs.
Of course, the presented Dantzig-Wolfe decomposition for linear or $0 - 1$ integer programs is just one type of column generation algorithms. Others solve the subordinate problem not via general linear or integer programming techniques, but use combinatorial or sort of explicit enumeration algorithms. Furthermore, the problem is often not modeled via (31), but directly as in (37). This is, for instance, the case when the set of feasible solutions have a rather complex description by linear

inequalities, but can easily be incorporated into some enumeration scheme.

The application area where Dantzig-Wolfe decomposition or column generation is used is very broad, for instance in airline crew scheduling, vehicle routing, public mass transport, network design, to name just a few. Of course, also integer programs with bordered block diagonal form, see Figure 1, nicely fit into this context. In contrast to Lagrangean relaxation, where the coupling constraints are relaxed, Dantzig-Wolfe decomposition keeps these constraints in the master problem and relaxes the constraints of the blocks having the advantage that (38) decomposes into independent problems, one for each block.

3.2.3 Benders' Decomposition

Let us finally turn to *Benders' decomposition* [10]. 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 dual to Dantzig-Wolfe decomposition, so one can say, Benders' reformulation is good in getting rid of complicating variables. Consider again (1) and write it in the form

$$\begin{aligned} \min \quad & c_1^T x_1 + c_2^T x_2 \\ \text{s.t.} \quad & A_1 x_1 + A_2 x_2 \le b \\ & x_1 \in \mathbb{R}^{n_1}, x_2 \in \mathbb{R}^{n_2}, \end{aligned} \tag{39}$$

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 (39) 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 below. 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 (39) to

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & -z + c_1^T x_1 + c_2^T x_2 \le 0 \\ & A_1 x_1 + A_2 x_2 \le b \\ & z \in \mathbb{R},\ x_1 \in \mathbb{R}^{n_1},\ x_2 \in \mathbb{R}^{n_2}, \end{aligned} \tag{40}$$

Now, (40) is equivalent to

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & -uz + uc_1^T x_1 + v^T A_1 x_1 \leq v^T b \\ & z \in \mathbb{R},\ x_1 \in \mathbb{R}^{n_1}, \\ & \binom{u}{v} \in C, \end{aligned} \tag{41}$$

where

$$C = \{\binom{u}{v} \in \mathbb{R}^{m+1} : v^T A_2 + uc_2^T = 0,\ u \geq 0, v \geq 0\}.$$

C is a polyhedral cone, thus there exist vectors $\binom{\bar{u}_1}{\bar{v}_1}, \ldots, \binom{\bar{u}_s}{\bar{v}_s}$ such that $C = \text{cone}(\{\binom{\bar{u}_1}{\bar{v}_1}, \ldots, \binom{\bar{u}_s}{\bar{v}_s}\})$. These extreme rays can be rescaled such that $\bar{u}_i$ is zero or one. Thus $C = \text{cone}(\{\binom{0}{v_k} : k \in K\}) + \text{cone}(\{\binom{1}{v_j} : j \in J\})$ with $K \cup J = \{1, \ldots, s\}$ and $K \cap J = \emptyset$. With this description of C, (41) can be restated as

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & -z \leq c_1^T x_1 + v_j^T (b - A_1 x_1) \quad \text{for all } j \in J, \\ & 0 \leq v_k^T (b - A_1 x_1) \quad \text{for all } k \in K, \\ & z \in \mathbb{R}, x_1 \in \mathbb{R}^{n_1}. \end{aligned} \tag{42}$$

(42) is called *Benders' master problem.* Benders' master problem has just $n_1 + 1$ variables instead of $n_1 + n_2$ variables in (39), or in case $x_1 \in \mathbb{Z}^{n_1}$ we have reduced the mixed integer program (39) to an almost pure integer program (42) with one additional continuous variable z. However, (42) 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. We start with a small subset of extreme rays of C (possibly the empty set) and optimize (42) 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 (42). This can be done via the following linear program

$$\begin{aligned} \min \quad & v^T (b - A_1 x_1^*) + u(z^* - c_1^T x_1^*) \\ \text{s.t.} \quad & \binom{u}{v} \in C. \end{aligned} \tag{43}$$

(43) is called *Benders' subproblem.* It is feasible, since $\binom{0}{0} \in C$, and (43) has an optimal solution value of zero or it is unbounded. In the first case, x_1^*, z^* satisfies all inequalities in (42) and we have solved (42) and thus (39). In the latter case we

obtain an extreme ray $\binom{u^*}{v^*}$ from (43) with $(v^*)^T(b - A_1 x_1^*) + u^*(z^* - c_1^T x_1^*) < 0$ which after rescaling yields a cut for (42) violated by x_1^*, z^*. We add this cut to Benders' master problem (42) and iterate.
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 application areas are problems whose constraint matrix has bordered block diagonal form, where we have coupling variables instead of coupling constraints, see Figure 3, i. e., the structure of the constraints is the transposed of the structure of the matrix in Figure 1. Such problems appear, for instance, in stochastic integer programming. Benders' decomposition is attractive in this case, because Benders' subproblem decomposes into k independent problems.

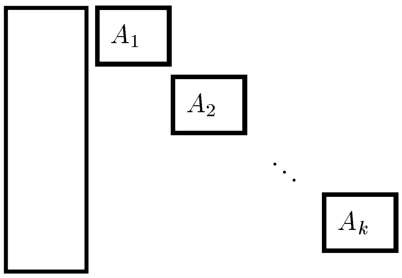

Figure 3: Matrix in bordered block diagonal form with coupling variables

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 touching certain (large) parts of the space of feasible solutions. Let $X := \{x \in \mathbb{Z}^{n-p} \times R^n : Ax \leq b\}$ be the set of feasible mixed integer solutions of problem (1). If it is too difficult to compute

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

(with a cutting plane approach, for instance,) 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

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

Afterwards 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 *branch-and-bound tree*. 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 we suppose we have chosen the LP relaxation method by relaxing the integrality constraints. It occasionally happens in the course of the branch-and-bound algorithm that the optimal solution $\tilde{x}_{X_{\tilde{j}}}$ of the LP relaxation of a subproblem $X_{\tilde{j}}$ is simultaneously a feasible mixed integer point, i. e., it lies in X. This allows us to maintain an upper bound $U := c^T \tilde{x}_{X_{\tilde{j}}}$ on the optimal solution value z_{MIP} of X, as $z_{\text{MIP}} \leq U$. To have a good upper bound U is crucial in a branch-and-bound algorithm, because it enables us to keep the branching tree small: Suppose the solution of relaxation of any 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 is in no way better than the best feasible solution $\tilde{x}_{X_{\tilde{j}}}$ 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 relaxation. Let* $\tilde{x}_{X_j}$ *be the optimal solution, so* $b_{X_j} := c^T \tilde{x}_{X_j}$.

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.*

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 the list 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 *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 relaxation. Let $\tilde{x}_{X_j}$ be the optimal solution, so $b_{X_j} := c^T \tilde{x}_{X_j}$.*
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.*
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 in improving the lower bound.*
7. *Split problem X_j into subproblems and add them to the list L.*
8. *Go to Step 2, until the list 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. Popular strategies are to branch on a variable that is closest to 0.5 and to choose a node with the worst dual bound. In this section we briefly discuss some more alternatives. We will see that they sometimes outperform the mentioned standard strategy. For a comprehensive study of branch-and-bound strategies we refer to [50, 52] and the references therein.

4.1 Node Selection

In the following 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 always one of the two sons are processed next, if the node could not be fathomed. This fact also implies that the linear programs from one node to the next are very similar, usually the difference is just the change of one variable bound and thus the reoptimization goes fast. The main disadvantage is that the dual bound basically stays untouched during the solution process resulting in bad solution guarantees.

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 [60] 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

$$\varrho(p) = z(p) + \frac{\bar{z}_{\text{IP}} - z(\text{root})}{s(\text{root})} \cdot s(p). \tag{44}$$

The term $\frac{\bar{z}_{\text{IP}} - z(\text{root})}{s(\text{root})}$ 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 [58] show that *dfs* finds by far the maximal 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 average much smaller (around one half) for *dfs* than using *bfs* or *best projection*. This confirms our statement that reoptimizing a linear program is fast when just one variable bound is changed. However, *dfs* forgets to work on the dual bound. For many more difficult problems the dual bound is not improved resulting in very bad solution guarantees 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 [52] 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 to add an inequality, but only to change 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, [16, 19, 61]. In the following we present three variable selection rules.

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

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 [11], we need some notation. Let $\mathcal{P}$ denote the set of all problems (nodes) except the root node that have already been solved in the solution process. Initially, this set is empty. $\mathcal{P}^+$ denotes the set of all right sons, and $\mathcal{P}^-$ the set of all left sons, where $\mathcal{P} = \mathcal{P}^+ \cup \mathcal{P}^-$. For some problem $p \in \mathcal{P}$ let

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

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

$$\Phi^+(j) = \frac{1}{|P_j^+|} \sum_{p \in P_j^+} \frac{z(p) - z(f(p))}{\lceil x_{v(p)}(f(p)) \rceil - x_{v(p)}(f(p))}, \tag{45}$$

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

$$\Phi^-(j) = \frac{1}{|P_j^-|} \sum_{p \in P_j^-} \frac{z(p) - z(f(p))}{x_{v(p)}(f(p)) - \lfloor x_{v(p)}(f(p)) \rfloor}, \tag{46}$$

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 variable j. There are many suggestions made on how to choose the sets P_j^+ and P_j^-, for a survey see [52]. To name one possibility, following the suggestion of Eckstein [26] 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

$$\Phi(j) = \alpha_j^+ \Phi^+(j) + \alpha_j^- \Phi^-(j), \tag{47}$$

where α_j^+, α_j^- are positive scalars. A variable that maximizes (47) is chosen to be the next branching variable. As formula (47) shows, the rule takes the previously obtained success of the variables into account when deciding on the next branching variable. The weakness of this approach is that at the very beginning there is no information available, and $\Phi(\cdot)$ is almost identical for all variables. Thus, at the beginning where the branching decisions are usually the most critical the pseudo-costs take no effect. This drawback is tried to overcome in the following rule.

Strong Branching. The idea of *strong branching*, invented by CPLEX [44] (see also [4]), 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 justification are based only on experimental results.

The computational experiences in [58] 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 in particular apparent if the number of 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 [52], 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.

4.3 Further Aspects

In the remainder of 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, [45, 24]. 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 application. 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 excess in the ongoing solution process.

Heuristics. Raising the lower bound using cutting planes is one important aspect in a branch-and-cut algorithm, finding good feasible solutions early to enable 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 alternatingly rounding/fixing some variables and solving linear programs, until all 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 in which the heuristics are called. Some more information hereto can be found, for instance, in [58, 21, 14].

Reduced Cost Fixing. The idea is to fix variables by exploiting the reduced costs of the current optimal LP solution. Let $\bar{z} = c^T\bar{x}$ be the objective function value of the current LP solution, z^{IP} be an upper bound on the value of the optimal solution, and $d = (d_i)_{i=1,\dots,n}$ the corresponding reduced cost vector. Consider a non-basic variable x_i of the current LP solution with finite lower and upper bounds l_i and u_i, and non-zero reduced cost d_i. Set $\delta = \frac{z^{\text{IP}} - \bar{z}}{|d_i|}$, 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 [22], in which case the variable can always be fixed if the criterion applies. There are problems where by the reduced cost criterion many variables can be fixed, see, for instance, [28]. 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 non-negative variables x_j, $j \in J$), all other variables in this SOS constraint can be fixed to zero.

5 Final Remarks

In this paper we 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 inherited 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 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 (2) into (3) if no additional information like 'this is a Steiner tree problem' is given. In other words, there are further tools necessary that allow to transmit such information. Modeling languages like `AMPL` [29] or `ZIMPL` [48] 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 semidefinite 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 with 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 [55].

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 got 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.

Software

The whole paper 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 view of them are business oriented. 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 [76].

The following list shows software where we know that it has included many of the aspects that are mentioned in this paper: `ABACUS`, developed at the University of Cologne [79], provides a branch-and-cut framework mainly for combinatorial optimization problems, `bc-opt`, developed at CORE [20], is very strong for mixed $0-1$ problems, `CPLEX`, developed at Incline Village [14, 45], is one of the currently best commercial codes, `MINTO`, developed at Georgia Institute of Technology [62], is excellent in cutting planes and has included basically all the mentioned cutting planes and more, `MIPO`, developed at Columbia University [7], is very good in lift-and-project cuts, `SIP` developed at Darmstadt University of Technology and ZIB, is the software of the authors, `SYMPHONY`, developed at Cornell University and Lehigh University [72], has its main focus on providing a parallel framework, and XPRESS-MP, developed at DASH [24], is also one of the best commercial codes.

References

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

[2] K. Aardal, R. Weismantel, and L.A. Wolsey. Non-standard approaches to integer programming. Technical Report CORE DP2000/2, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 2000.

[3] E.D. Andersen and K.D. Andersen. Presolving in linear programming. *Mathematical Programming*, 71:221 – 245, 1995.

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

[5] E. Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8:146 – 164, 1975.

[6] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed $0-1$ programs. *Mathematical Programming*, 58:295–324, 1993.

[7] E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42:1229 – 1246, 1996.

[8] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1 – 9, 1996.

[9] E. Balas and E. Zemel. Facets of the knapsack polytope from minimal covers. *SIAM Journal on Applied Mathematics*, 34:119 – 148, 1978.

[10] J.F. Benders. Partitioning procedures for solving mixed variables programming. *Numerische Mathematik*, 4:238–252, 1962.

[11] M. Benichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer programming. *Mathematical Programming*, 1:76 – 94, 1971.

[12] R.E. Bixby. *Lectures on Linear Programming*. Rice University, Houston, Texas, Spring 1994.

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

[14] R.E. Bixby, M. Fenelon, Z. Guand E. Rothberg, and R. Wunderling. MIP: Theory and practice closing the gap. Technical report, ILOG Inc., Paris, France, 1999.

[15] R. Borndörfer. *Aspects of Set Packing, Partitioning, and Covering*. Shaker, Aachen, 1998.

[16] R. Borndörfer, C.E. Ferreira, and A. Martin. Decomposing matrices into blocks. *SIAM Journal on Optimization*, 9:236 – 269, 1998.

[17] S. Ceria, C. Cordier, H. Marchand, and L.A. Wolsey. Cutting planes for integer programs with general integer variables. *Mathematical Programming*, 81:201 – 214, 1998.

[18] S. Chopra and M.R. Rao. The Steiner tree problem I: Formulations, compositions and extension of facets. *Mathematical Programming*, 64:209 – 229, 1994.

[19] J.M. Clochard and D. Naddef. Using path inequalities in a branch-and-cut code for the symmetric traveling salesman problem. In L.A. Wolsey and G. Rinaldi, editors, *Proceedings on the Third IPCO Conference*, pages 291–311, 1993.

[20] C. Cordier, H. Marchand, R. Laundy, and L.A. Wolsey. bc – opt: a branch-and-cut code for mixed integer programs. Technical Report CORE DP9778, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 1997.

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

[22] H. Crowder, E. Johnson, and M.W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.

[23] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

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

[25] I.R. de Farias, E.L. Johnson, and G.L. Nemhauser. Facets of the complementarity knapsack polytope. Technical Report LEC-98-08, Georgia Institute of Technology, 1998.

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

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

[28] C.E. Ferreira, A. Martin, and R. Weismantel. Solving multiple knapsack problems by cutting planes. *SIAM Journal on Optimization*, 6:858 – 877, 1996.

[29] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press / Brooks/Cole Publishing Company, 1993.

[30] D.R. Fulkerson. Blocking and Anti-Blocking Pairs of Polyhedra. *Mathematical Programming*, 1:168–194, 1971.

[31] R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society*, 64:275 – 278, 1958.

[32] R.E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The RAND Cooperation, 1960.

[33] R.E. Gomory. Solving linear programming problems in integers. In R. Bellman and M. Hall, editors, *Combinatorial analysis, Proceedings of Symposia in Applied Mathematics*, volume 10, Providence RI, 1960.

[34] J. Gondzio. Presolve analysis of linear programs prior to apply an interior point method. *INFORMS Journal on Computing*, 9:73 – 91, 1997.

[35] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

[36] M. Grötschel, C.L. Monma, and M. Stoer. Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations Research*, 40:309 – 330, 1992.

[37] Z. Gu, G.L. Nemhauser, and M.W.P. Savelsbergh. Cover inequalities for $0-1$ linear programs: complexity. *INFORMS Journal on Computing*, 11:117 – 123, 1998.

[38] Z. Gu, G.L. Nemhauser, and M.W.P. Savelsbergh. Cover inequalities for $0-1$ linear programs: computation. *INFORMS Journal on Computing*, 10:427 – 437, 1998.

[39] Z. Gu, G.L. Nemhauser, and M.W.P. Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. *Mathematical Programming*, 85:439 – 468, 1999.

[40] Z. Gu, G.L. Nemhauser, and M.W.P. Savelsbergh. Sequence independent lifting in mixed integer programming. *Journal on Combinatorial Optimization*, 4:109 – 129, 2000.

[41] P.L. Hammer, E.L. Johnson, and U.N. Peled. Facets of regular 0-1 polytopes. *Mathematical Programming*, 8:179 – 206, 1975.

[42] M. Held and R. Karp. The traveling-salesman problem and minimum spanning trees: Part ii, 1971.

[43] K.L. Hoffman and M.W. Padberg. Improved LP-representations of zero-one linear programs for branch-and-cut. *ORSA Journal on Computing*, 3:121–134, 1991.

[44] ILOG CPLEX Division, 889 Alder Avenue, Suite 200, Incline Village, NV 89451, USA. *Using the CPLEX Callable Library*, 1997. Information available at URL http://www.cplex.com.

[45] ILOG CPLEX Division, 889 Alder Avenue, Suite 200, Incline Village, NV 89451, USA. *Using the CPLEX Callable Library*, 2000. Information available at URL http://www.cplex.com.

[46] E. Johnson and M.W. Padberg. A note on the knapsack problem with special ordered sets. *Operations Research Letters*, 1:18 – 22, 1981.

[47] D. Klabjan, G.L. Nemhauser, and C. Tovey. The complexity of cover inequality separation. *Operations Research Letters*, 23:35 – 40, 1998.

[48] T. Koch. ZIMPL user guide. Technical Report Preprint 01-20, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2001.

[49] T. Koch, A. Martin, and S. Voß. SteinLib: An updated library on Steiner tree problems in graphs. Preprint Nr. 2135, Darmstadt University of Technology, Department of Mathematics, December 2000. To appear in *Steiner trees in Industry*.

[50] A. Land and S. Powell. Computer codes for problems of integer programming. *Annals of Discrete Mathematics*, 5:221 – 269, 1979.

[51] C. Lemaréchal and A. Renaud. A geometric study of duality gaps, with applications. *Mathematical Programming*, 90:399 – 427, 2001.

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

[53] L. Lovász and A. Schrijver. Cones of matrices and set-functions and $0-1$ optimization. *SIAM Journal on Optimization*, 1:166 – 190, 1991.

[54] H. Marchand. *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, 1998.

[55] H. Marchand, A. Martin, R. Weismantel, and L.A. Wolsey. Cutting planes in integer and mixed integer programming. Technical Report CORE DP9953, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 1999.

[56] H. Marchand and L.A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. Technical Report CORE DP9839, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 1998.

[57] H. Marchand and L.A. Wolsey. The $0-1$ knapsack problem with a single continuous variable. *Mathematical Programming*, 85:15 – 33, 1999.

[58] A. Martin. Integer programs with block structure. Habilitations-Schrift, Technische Universität Berlin, 1998.

[59] A. Martin and R. Weismantel. The intersection of knapsack polyhedra and extensions. In R.E. Bixby, E.A. Boyd, and R.Z.Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, Proceedings of the 6th IPCO Conference, pages 243 – 256, 1998.

[60] G. Mitra. Investigations of some branch and bound strategies for the solution of mixed integer linear programs. *Mathematical Programming*, 4:155 – 170, 1973.

[61] D. Naddef. Polyhedral theory and branch-and-cut algorithms for the symmetric tsp. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*. Kluwert, 2001. To appear.

[62] G.L. Nemhauser, M.W.P. Savelsbergh, and G.C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47 – 58, 1994.

[63] G.L. Nemhauser and P. H. Vance. Lifted cover facets of the $0-1$ knapsack polytope with GUB constraints. *Operations Research Letters*, 16:255 – 263, 1994.

[64] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

[65] G.L. Nemhauser and L.A. Wolsey. A recursive procedure to generate all cuts for $0-1$ mixed integer programs. *Mathematical Programming*, 46:379 – 390, 1990.

[66] M.W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[67] M.W. Padberg. A note on zero-one programming. *Operations Research*, 23:833–837, 1975.

[68] M.W. Padberg. $(1, k)$-configurations and facets for packing problems. *Mathematical Programming*, 18:94–99, 1980.

[69] M.W. Padberg. Classical cuts for mixed-integer programming and branch-and-cut. Technical report, New York University, 2000. To appear in MMOR.

[70] M.W. Padberg, T.J. Van Roy, and L.A. Wolsey. Valid inequalities for fixed charge problems. *Operations Research*, 33:842 – 861, 1985.

[71] Y. Pochet. Valid inequalities and separation for capacitated economic lot-sizing. *Operations Research Letters*, 7:109 – 116, 1988.

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

[73] T.J. Van Roy and L.A. Wolsey. Valid inequalities for mixed 0-1 programs. *Discrete Applied Mathematics*, 4:199 – 213, 1986.

[74] T.J. Van Roy and L.A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35:45 – 57, 1987.

[75] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, 1986.

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

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

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

[79] S. Thienel. *ABACUS A Branch-And-CUt System*. PhD thesis, Universität zu Köln, 1995.

[80] R. Weismantel. On the 0/1 knapsack polytope. *Mathematical Programming*, 77:49–68, 1997.

[81] L.A. Wolsey. Faces of linear inequalities in 0-1 variables. *Mathematical Programming*, 8:165 – 178, 1975.

[82] L.A. Wolsey. Valid inequalities for 0-1 knapsacks and MIPs with generalized upper bound constraints. *Discrete Applied Mathematics*, 29:251–261, 1990.

[83] E. Zemel. Easily computable facets of the knapsack polytope. *Mathematics of Operations Research*, 14:760 – 764, 1989.