General Mixed Integer Programming: Computational Issues for Branch-and-Cut Algorithms

Alexander Martin

Darmstadt University of Technology, Department of Mathematics, Schlossgartenstr. 7, D-64289 Darmstadt, Germany

Abstract. In this paper we survey the basic features of state-of-the-art branch-and-cut algorithms for the solution of general mixed integer programming problems. In particular we focus on preprocessing techniques, branch-and-bound issues and cutting plane generation.

1 Introduction

A general mixed integer program (MIP) is the problem of optimizing a linear objective function subject to a system of linear constraints, where some or all of the variables are required to be integer. The solution of general mixed integer programs is one of the challenging problems in discrete optimization. The problems that can be modeled as mixed integer programs arise, for instance, in science, technology, business, and environment, and their number is tremendous. It is therefore no wonder that many solution methods and codes exist for the solution of mixed integer programs, and not just a few of them are business oriented, see [63] for a survey on commercial linear and integer programming solvers.

One of the most successful methods to solve mixed integer programming problems are branch-and-cut algorithms. In Section 2 we outline the principle structure of a branch-and-cut algorithm. The main ingredients are preprocessing, the solution of the underlying linear programs, branch-and-bound issues, cut generation, and primal heuristics. Our aim is to give sort of a survey on the features that state-of-the-art branch-and-cut solvers for mixed integer programs include. Most of the issues presented are pretty much standard, but our intention is to use this paper more as a text book and to give the unfamiliar reader of this subject an impression on how mixed integer programs are solved today. In detail we focus on preprocessing in Section 3, branch-and-bound issues in Section 4, and cut generation in Section 5.

The software package that we use as a basic reference in this paper is `SIP`, which is currently developed at our institute and ZIB [48]. As mentioned most of the described issues are common to basically all state-of-the-art solvers and there are many other comparable codes that contain many of the described features. Among them are in particular `ABACUS`, developed at the University of Cologne [65], `bc-opt`, developed at CORE [19], `CPLEX`, developed at ILOG [39], `MIPO`, developed at Columbia University [6], `MINTO`, developed at Georgia Institute of Technology [52], `SYMPHONY`, developed at Cornell University and Lehigh University (see also the article of Leslie Trotter and Ted Ralphs in this book), and `XPRESS-MP`, developed at DASH [21].

It is common to use the library `Miplib` [10] as test set to evaluate certain features of a MIP code. `Miplib` is a collection of real-world mixed integer programming problems. From time to time we will refer to some instances from this library to explain certain phenomena. However, we will not give computational results here. The reader interested in concrete running times will find them, for instance, in [11, 19, 44, 48].

Of course, branch-and-cut algorithms are not the only successful way to solve general mixed integer programs. For an excellent survey on alternative approaches, including test sets, Gomory's group approach and basis reduction, see [1].

2 Branch-and-Cut Algorithms

In this section we sketch the main ideas of a branch-and-cut algorithm. More details and references on this subject can be found in the survey article [14]. Suppose we want to solve a mixed integer

program

$$\begin{aligned} &\min\ c^T x \\ &\text{s.t.}\quad Ax \le b, \end{aligned} \tag{1}$$

where $A \in \mathbb{Q}^{m\times n}$, $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$; the variables $x_i\,(i = 1,\dots,n)$ might be binary ($x_i \in \{0,1\}$), integer ($x_i \in \mathbb{Z}$), or continuous ($x_i \in \mathbb{R}$). Let $P_{\text{IP}} = \text{conv}\{x \in \mathbb{R}^n : x \text{ is feasible for (1)}\}$. The first step of the algorithm is to consider a relaxation of (1) by choosing a set $P' \subseteq \mathbb{R}^n$ with $P_{\text{IP}} \subseteq P'$ and to optimize the linear objective function over P'. For example, this relaxation might be the linear programming relaxation $\min\{c^T x : Ax \le b\}$ or a semidefinite relaxation. We only consider linear relaxations, hence, the set P' is always a polyhedron.

Let $\bar{x}$ be an optimal solution for the linear relaxation. If $\bar{x}$ is integer and all inequalities of $Ax \le b$ are satisfied by $\bar{x}$, we have found an optimal solution for (1). Otherwise, there exists a hyperplane $\{x \in \mathbb{R}^n : a^T x = \alpha\}$ such that $a^T\bar{x} > \alpha$ and $P_{\text{IP}} \subseteq \{x \in \mathbb{R}^n : a^T x \le \alpha\}$. Such a hyperplane is called a *cutting plane*. The problem of finding such a hyperplane is called the *separation problem*. More precisely,

> given $\bar{x} \in \mathbb{R}^n$. Decide, whether $\bar{x} \in P_{\text{IP}}$. If not, find some valid inequality $a^T x \le \alpha$ for P_{IP} such that $a^T\bar{x} > \alpha$.

It is well known that the separation problem for P_{IP} and the optimization problem $\min\{c^T x : x \in P_{\text{IP}}\}$ are polynomially equivalent, see [30, 31]. Sometimes, the separation problem is restricted to a certain class of inequalities, in which case we are searching for a violated inequality of that class. If we are able to find such a cutting plane, we can strengthen the relaxation and continue, for details see Section 5. This process is iterated until $\bar{x}$ is a feasible solution or no more violated inequalities are found. In the latter case this so-called *cutting plane phase* is embedded into an enumeration scheme. This is commonly done by picking some fractional variable $\bar{x}_i$ that must be binary or integer and creating two subproblems, one where one requires $x_i \ge \lceil\bar{x}_i\rceil$, and one where $x_i \le \lfloor\bar{x}_i\rfloor$, see also the discussions in Section 4. The following algorithm summarizes the whole procedure.

Algorithm 1. (Branch-and-Cut Algorithm)

(1) Let L be a list of unsolved problems. Initialize L with (1).
(2) **Repeat**
(3) Choose a problem Π from L and delete it from L.
(4) **Repeat** *(iterate)*
(5) Solve the (linear) relaxation of Π. Let $\bar{x}$ be an optimal solution.
(6) If $\bar{x}$ is feasible for Π, Π is solved; **goto** (10).
(7) Look for violated inequalities and add them to the LP.
(8) **Until** there are no violated inequalities
(9) Split Π into subproblems and add them to L.
(10) **Until** $L = \emptyset$.
(11) Print the optimal solution.
(12) **STOP.**

The list L is usually organized as a binary tree, the so-called *branch-and-bound tree*. Each (sub)problem Π corresponds to a node in the tree, where the unsolved problems are the *leaves* of the tree and the node that corresponds to the entire problem (1) is the *root*. In the remainder of this section we discuss some issues that can be found in basically every state-of-the-art branch-and-cut implementation.

LP-Management. We assume that the reader is familiar with linear programming techniques. A comprehensive treatment of this subject can be found in [17, 58]. 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, [39, 21].

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 [48, 19, 11].

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 [20], 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, [24]. 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.

Enumeration Aspects. In our description of a branch-and-cut algorithm we left the questions open which problem to choose in Step (3) and how to split the problem in Step (9). We discuss these issues in detail in Section 4.

3 Preprocessing

Before entering the branch-and-cut phase as outlined in Section 2 there is usually a preprocessing step prefixed. Preprocessing aims at eliminating redundant information from the problem formulation given by the user and simultaneously tries to strengthen the formulation by logical implications. Preprocessing can be very effective and sometimes it might not be possible to solve certain problems without a good preprocessing. This includes, for instance, Steiner tree problems [42] or set partitioning problems [12]. Typically, preprocessing is applied only once at the beginning of the solution procedure, but sometimes it pays to run the preprocessing routine more often on different nodes in the branch-and-bound phase, see, for instance, [12, 37, 11]. There is always the question of the break even point between the running time for preprocessing and the savings in the solution

time for the whole problem. There is no unified answer to this question. It depends on the individual problem, when intensive preprocessing pays and when not. In the following we discuss some preprocessing options and ways to implement them. Most of these options are incorporated in our code SIP and are drawn from [2, 9, 20, 37, 64].

We extend our definition of a mixed integer program in (1) slightly and consider it in the following more general form:

$$\begin{aligned} &\min c^T x \\ &\text{s.t. } Ax \left\{ \begin{matrix} \leq \\ = \\ \geq \end{matrix} \right\} b \\ &\quad l \leq x \leq u \\ &\quad x \in \mathbb{Z}^N \times \mathbb{R}^C, \end{aligned} \tag{2}$$

where M, N, and C are finite sets with N and C disjoint, $A \in \mathbb{R}^{M \times (N \cup C)}$, $c, l, u \in \mathbb{R}^{N \cup C}$, $b \in \mathbb{R}^M$. If some variable x_i, $i \in N$, is binary we have $l_i = 0$ and $u_i = 1$. If some variable x_j has no upper or lower bound, we assume that $l_j = -\infty$ or $u_j = +\infty$. Again we define $P_{\text{IP}} = \text{conv}\{x \in \mathbb{R}^n : x \text{ is feasible for (2)}\}$. Furthermore, we denote by $s_i \in \{\leq, =, \geq\}$ the sign of row i, i. e., (2) reads $\min\{c^T x : Ax \, s \, b, \, l \leq x \leq u, \, x \in \mathbb{Z}^N \times \mathbb{R}^C\}$. In order to avoid too many subcases in the following discussion we assume without loss of generality that there are no "greater than or equal" inequalities, i. e., $s_i \in \{\leq, =\}$. We consider the following cases:

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

$$s_i = \left\{ \begin{matrix} \leq \\ = \end{matrix} \right\} \text{ and } \left\{ \begin{matrix} b_i < 0 \\ |b_i| > 0 \end{matrix} \right\}$$

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

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

$$l_j > u_j,$$

the problem is infeasible. If

$$u_j = l_j,$$

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

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

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

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

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

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

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

(d) s_i = '$\leq$', s_j = '$\leq$', and $\alpha b_i > b_j$ ($\alpha > 0$)
In the first three cases row j is redundant, in (2d) row i.

3. Range constraints:
   (a) s_i = '$\leq$', s_j = '$\leq$', and $\alpha b_i = b_j$ ($\alpha < 0$)
   The two inequalities can be aggregated into one equation.
   (b) s_i = '$\leq$', s_j = '$\leq$', and $\alpha b_i < b_j$ ($\alpha < 0$)
   In this case both inequalities can be aggregated into one range constraint of the form $A_i.x + u = b_i$ with $0 \leq u \leq b_i - \frac{b_j}{\alpha}$.

The question remains how to find parallel rows. Tomlin and Welsh [66] describe an efficient procedure, when the matrix A is stored columnwise, and Andersen and Andersen [2] slightly refine this approach. The idea is to use a hash function such that rows in different baskets are not parallel. Possible hash functions are the number of non-zeros, the index of the first and/or the last index of the row, the coefficient of the first non-zero entry, etc. In practice, the baskets are rather small so that rows inside one basket can be checked pairwise.

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

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

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

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

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

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

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

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

$$\begin{aligned} L_i &= \sum_{j \in P_i} a_{ij} l_j + \sum_{j \in N_i} a_{ij} u_j \\ U_i &= \sum_{j \in P_i} a_{ij} u_j + \sum_{j \in N_i} a_{ij} l_j \end{aligned} \tag{3}$$

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`):

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

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

$$\text{If} \begin{cases} a_{ij} < 0, U_i + a_{ij} < b_i, \text{ set} & a'_{ij} = b_i - U_i, \\ a_{ij} > 0, U_i - a_{ij} < b_i, \text{ set} & \begin{cases} a'_{ij} = U_i - b_i, \\ b_i = U_i - a_{ij}, \end{cases} \end{cases} \tag{4}$$

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

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

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

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

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

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

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

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

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

$$\frac{b_i - a_{ij}x_j}{a_{ik}} \,. \tag{5}$$

In case x_k is binary or integer, the substitution is only possible, if the term (5) 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 [29].
On the other hand, disaggregation of rows is an important issue for mixed integer programs. Consider the following inequality (taken from the `Miplib`-problem `p0282`)

$$x_{85} + x_{90} + x_{95} + x_{100} + x_{217} + x_{222} + x_{227} + x_{232} - 8x_{246} \leq 0 \tag{6}$$

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 (6) by the following eight inequalities:

$$x_i - x_{246} \leq 0 \qquad \text{for all } i \in S. \tag{7}$$

Concerning the LP-relaxation, this formulation is tighter. Whenever any variable in S is one, x_{246} is forced to one as well, which is not guaranteed in the original formulation. On the other hand, one constraint is replaced by many (in our case 8) inequalities, which might blow up the constraint matrix. However within a cutting plane procedure this problem is not really an issue, because the inequalities in (7) can be generated on demand.

Probing. Probing is sometimes used in general mixed integer programming codes, see, for instance, [64]. 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:

$$\begin{aligned} (x_j = 1 \Rightarrow x_i = \alpha) &\Rightarrow \begin{cases} x_i \geq l_i + (\alpha - l_i)x_j \\ x_i \leq u_i - (u_i - \alpha)x_j \end{cases} \\ (x_j = 0 \Rightarrow x_i = \alpha) &\Rightarrow \begin{cases} x_i \geq \alpha - (\alpha - l_i)x_j \\ x_i \leq \alpha + (u_i - \alpha)x_j \end{cases} \end{aligned} \tag{8}$$

As an example, suppose we set in (6) variable x_{246} temporary to zero. This implies that $x_i = 0$ for all $i \in S$. Applying (8) we deduce the inequality

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

for all $i \in S$ which is exactly (7).

In general, all these tests are iteratively applied until all of them fail. In other words, the original formulation is strengthened as far as possible. Our computational experiences [48] show that presolve reduces the problem sizes in terms of number of rows, columns, and non-zeros by around 10%. The time spent in presolve is neglectable (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.

4 Branch-and-Bound Strategies

In the general outline of a branch-and-cut algorithm, see Algorithm 1 in Section 2, there are two steps in the branch-and-bound part that leave some choices. In Step (3) of Algorithm 1 we have to select the next problem (node) from the list of unsolved problems to work on next, and in Step (9) we must decide on how to split the problem into subproblems. Popular strategies are to branch on a variable that is closest to 0.5 and to choose a node with the worst dual bound. In this section we briefly discuss some more alternatives. We will see that they sometimes outperform the mentioned standard strategy. For a comprehensive study of branch-and-bound strategies we refer to [43, 44] and the references therein. We assume in this section that a general mixed integer program of the form (2) is given.

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 1.

1. Best First Search (bfs).
   Here, a node is chosen with the worst dual bound, i. e., a node with lowest lower bound, since we are minimizing in (2). The goal is to improve the dual bound. However, if this fails early in the solution process, the branch-and-bound tree tends to grow considerably resulting in large memory requirements.
2. Depth First Search (dfs).
   This rule chooses the node that is "deepest" in the branch-and-bound tree, i. e., whose path to the root is longest. The advantages are that the tree tends to stay small, since always one of the two sons are processed next, if the node could not be fathomed. This fact also implies that the linear programs from one node to the next are very similar, usually the difference is just the change of one variable bound and thus the reoptimization goes fast. The main disadvantage is that the dual bound basically stays untouched during the solution process resulting in bad solution guarantees.
3. Best Projection.
   When selecting a node the 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 [50] 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

$$\varrho(p) = z(p) + \frac{\bar{z}_{\text{IP}} - z(\text{root})}{s(\text{root})} \cdot s(p). \tag{9}$$

   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 [48] 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 [44] 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 (9) of Algorithm 1. 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, [13, 18, 51]. In the following we present three variable selection rules.

1. Most Infeasibility.
   This rule chooses the variable that is closest to 0.5. The heuristic reason behind this choice is that this is a variable where the least tendency can be recognized to which "side" (up or down) the variable should be rounded. The hope is that a decision on this variable has the greatest impact on the LP relaxation.
2. Pseudo-costs.
   This is a more sophisticated rule in the sense that it keeps a history of the success of the variables on which one has already branched. To introduce this rule, which goes back to [8], 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

$$\Phi^+(j) = \frac{1}{|P_j^+|} \sum_{p \in P_j^+} \frac{z(p) - z(f(p))}{\lceil x_{v(p)}(f(p)) \rceil - x_{v(p)}(f(p))}, \tag{10}$$

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

$$\Phi^-(j) = \frac{1}{|P_j^-|} \sum_{p \in P_j^-} \frac{z(p) - z(f(p))}{x_{v(p)}(f(p)) - \lfloor x_{v(p)}(f(p)) \rfloor}, \tag{11}$$

   where $P_j^- \subseteq \mathcal{P}^-$. The terms $\frac{z(p)-z(f(p))}{\lceil x_{v(p)}(f(p)) \rceil - x_{v(p)}(f(p))}$ and $\frac{z(p)-z(f(p))}{x_{v(p)}(f(p)) - \lfloor x_{v(p)}(f(p)) \rfloor}$, respectively, measure the change in the objective function per unit decrease of infeasibility of variable j. There are many suggestions made on how to choose the sets P_j^+ and P_j^-, for a survey see [44]. To name one possibility, following the suggestion of Eckstein [22] 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

$$\Phi(j) = \alpha_j^+ \Phi^+(j) + \alpha_j^- \Phi^-(j), \tag{12}$$

   where α_j^+, α_j^- are positive scalars. A variable that maximizes (12) is chosen to be the next branching variable. As formula (12) 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.

3. Strong Branching.
   The idea of *strong branching*, invented by CPLEX [38] (see also [3]), 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 [48] 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 [44], 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.

5 Cutting Planes

In this section we discuss cutting planes known from the literature that are incorporated in general MIP solvers. Cutting planes for integer programs may be classified with regard to the question whether their derivation requires knowledge about the structure of the underlying constraint matrix. In Section 5.1 we describe cutting planes that do not exploit any structure. An alternative approach to obtain cutting planes for a mixed integer program follows essentially the scheme to derive relaxations associated with certain substructures of the underlying constraint matrix, and tries to find valid inequalities for these relaxations. Crowder, Johnson and Padberg [20] pioneered this methodology by interpreting each single row of the constraint matrix as a knapsack relaxation and strengthening the integer program by adding violated knapsack inequalities. This will be the topic of Section 5.2.

5.1 Cutting Planes Independent of any Problem Structure

Examples of families of cutting planes that do not exploit the structure of the constraint matrix are mixed integer Gomory cuts [25, 27, 28, 16, 62], mixed integer rounding cuts [54], and lift-and-project cuts [5]. Marchand [45] describes the merits of applying mixed integer rounding cuts, see also the article by Yves Pochet in this book. Lift-and-project cuts are investigated in [5, 6] and are comprehensively discussed in the article by Egon Balas in this book.

In this section we concentrate on Gomory's mixed integer cuts. As a warm-up we start with the pure integer case. We will see that this approach (based on a rounding argument) fails if continuous variables are involved. In the general mixed integer case a disjunctive argument saves us.

Pure Integer Programs

Consider a pure integer program in the form $\min\{c^T x : Ax = b, x \in \mathbb{Z}^n_+\}$ with A, b integer. Set $P_{\text{IP}} = \text{conv}\{x \in \mathbb{Z}^n_+ : Ax = b\}$. Let $\bar{x}$ be an optimal solution of the LP relaxation $\min\{c^T x : x \in P\}$ with $P = \{x \in \mathbb{R}^n_+ : Ax = b\}$ and $B \subseteq \{1, \ldots, n\}$ be a basis of A with $\bar{x}_B = A_B^{-1} b - A_B^{-1} A_N x_N$ and $\bar{x}_N = 0$, where $N = \{1, \ldots, n\} \setminus B$.

If $\bar{x}$ is integer, we terminate with an optimal solution for $\min\{c^T x : x \in P_{\text{IP}}\}$. Otherwise, one of the values $\bar{x}_B$ must be fractional. Let $i \in B$ be some index with $\bar{x}_i \notin \mathbb{Z}$. Since every feasible integral solution $x \in P_{\text{IP}}$ satisfies $x_B = A_B^{-1} b - A_B^{-1} A_N x_N$,

$$A_{i\cdot}^{-1} b - \sum_{j \in N} A_{i\cdot}^{-1} A_{\cdot j} x_j \in \mathbb{Z}. \tag{13}$$

The term on the left remains integral when adding integer multiples of x_j, $j \in N$, or an integer to $A_{i\cdot}^{-1} b$. We obtain

$$f(A_{i\cdot}^{-1} b) - \sum_{j \in N} f(A_{i\cdot}^{-1} A_{\cdot j}) x_j \in \mathbb{Z}, \tag{14}$$

where $f(\alpha) = \alpha - \lfloor \alpha \rfloor$, for $\alpha \in \mathbb{R}$. Since $0 \le f(\cdot) < 1$ and $x \ge 0$, we conclude that

$$f(A_{i\cdot}^{-1} b) - \sum_{j \in N} f(A_{i\cdot}^{-1} A_{\cdot j}) x_j \le 0,$$

or equivalently,

$$\sum_{j \in N} f(A_{i\cdot}^{-1} A_{\cdot j}) x_j \ge f(A_{i\cdot}^{-1} b) \tag{15}$$

is valid for P_{IP}. Moreover, it is violated by the current linear programming solution $\bar{x}$, since $\bar{x}_N = 0$ and $f(A_{i\cdot}^{-1} b) = f(\bar{x}_i) > 0$. After subtracting $x_i + \sum_{j \in N} A_{i\cdot}^{-1} A_{\cdot j} x_j = A_{i\cdot}^{-1} b$ from (15) we obtain

$$x_i + \sum_{j \in N} \lfloor A_{i\cdot}^{-1} A_{\cdot j} \rfloor x_j \le \lfloor A_{i\cdot}^{-1} b \rfloor, \tag{16}$$

which is, when the right-hand side is not rounded, a supporting hyperplane with integer left-hand side. Moreover, adding this inequality to the system $Ax = b$ preserves the property that all data are integral. Thus, the slack variable that is to be introduced for the new inequality can be required to be integer as well and the whole procedure can be iterated. In fact, Gomory [28] proves that with a particular choice of the generating row such cuts lead to a finite algorithm, i. e., after adding a finite number of inequalities, an integer optimal solution is found.

Later Chvátal [16, 62] found a distinct but closely related way of finding a linear description of P_{IP}. He showed when using all supporting hyperplanes with integer left-hand side (an example of such an hyperplane is given in (16)) and rounding the right-hand sides yields again a polyhedron that contains P_{IP}. In addition, he proved that iterating this process a finite number of times provides P_{IP}.

Mixed Integer Programs

The two approaches discussed so far fail when both integer and continuous variables are present. Chvátal's approach fails because the right-hand side of a supporting hyperplane cannot be rounded down. Gomory's approach fails since it is no longer possible to add integer multiples to continuous variables to derive (14) from (13). For instance, $\frac{1}{3} + \frac{1}{3} x_1 - 2x_2 \in \mathbb{Z}$ with $x_1 \in \mathbb{Z}_+, x_2 \in \mathbb{R}_+$ has a larger solution set than $\frac{1}{3} + \frac{1}{3} x_1 \in \mathbb{Z}$. As a consequence, we cannot guarantee that the coefficients of the continuous variables are non-negative and therefore show the validity of (15). Nevertheless, it is possible to derive valid inequalities using the following *disjunctive argument*.

Property 1. Let $(a^k)^T x \leq \alpha^k$ be a valid inequality for a polyhedron P^k for $k = 1, 2$. Then,

$$\sum_{i=1}^{n} \min(a_i^1, a_i^2) x_i \leq \max(\alpha^1, \alpha^2)$$

is valid for both $P^1 \cup P^2$ and $\text{conv}(P^1 \cup P^2)$.

This property applied in different ways yields valid inequalities for the mixed integer case. We present Gomory's mixed integer cuts here, the other two, mixed integer rounding cuts and lift-and-project-cuts are both more or less also based on Property 1, see the corresponding articles in this book.

Consider again the situation in (13), where $x_i, i \in B$, is required to be integer. We use the following abbreviations $\bar{a}_j = A_{i\cdot}^{-1} A_{\cdot j}$, $\bar{b} = A_{i\cdot}^{-1} b$, $f_j = f(\bar{a}_j)$, $f_0 = f(\bar{b})$, and $N^+ = \{j \in N : \bar{a}_j \geq 0\}$ and $N^- = N \setminus N^+$. Expression (13) is equivalent to $\sum_{j \in N} \bar{a}_j x_j = f_0 + k$ for some $k \in \mathbb{Z}$. We distinguish two cases, $\sum_{j \in N} \bar{a}_j x_j \geq f_0 \geq 0$ and $\sum_{j \in N} \bar{a}_j x_j \leq f_0 - 1 < 0$. In the first case,

$$\sum_{j \in N^+} \bar{a}_j x_j \geq f_0$$

must hold. In the second case, we have $\sum_{j \in N^-} \bar{a}_j x_j \leq f_0 - 1$, which is equivalent to

$$-\frac{f_0}{1 - f_0} \sum_{j \in N^-} \bar{a}_j x_j \geq f_0.$$

Now we apply Property 1 to the disjunction $P^1 = P_{\text{IP}} \cap \{x : \sum_{j \in N} \bar{a}_j x_j \geq 0\}$ and $P^2 = P_{\text{IP}} \cap \{x : \sum_{j \in N} \bar{a}_j x_j \leq 0\}$ and obtain the valid inequality

$$\sum_{j \in N^+} \bar{a}_j x_j - \frac{f_0}{1 - f_0} \sum_{j \in N^-} \bar{a}_j x_j \geq f_0. \tag{17}$$

This inequality may be strengthened in the following way. Observe that the derivation of (17) remains unaffected when adding integer multiples to integer variables. By doing this we may put each integer variable either in the set N^+ or N^-. If a variable is in N^+, the final coefficient in (17) is $\bar{a}_j$ and thus the best possible coefficient after adding integer multiples is $f_j = f(\bar{a}_j)$. In N^- the final coefficient in (17) is $-\frac{f_0}{1-f_0}\bar{a}_j$ and thus $\frac{f_0(1-f_j)}{1-f_0}$ is the best choice. Overall, we obtain the best possible coefficient by using $\min(f_j, \frac{f_0(1-f_j)}{1-f_0})$. This yields Gomory's mixed integer cut [26]

$$\begin{aligned} \sum_{\substack{j:\, f_j \leq f_0 \\ j \text{ integer}}} f_j x_j + \sum_{\substack{j:\, f_j > f_0 \\ j \text{ integer}}} \tfrac{f_0(1-f_j)}{1-f_0} x_j + \\ \sum_{\substack{j \in N^+ \\ j \text{ non-integer}}} \bar{a}_j x_j - \sum_{\substack{j \in N^- \\ j \text{ non-integer}}} \tfrac{f_0}{1-f_0} \bar{a}_j x_j \geq f_0. \end{aligned} \tag{18}$$

Gomory [26] shows that an algorithm based on iteratively adding these inequalities solves $\min\{c^T x : x \in X\}$ with $X = \{x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p} : Ax = b\}$ in a finite number of steps provided $c^T x \in \mathbb{Z}$ for all $x \in X$.

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 [59]. In [7, 11] it is shown how useful Gomory cuts are if they are incorporated in the right way.

5.2 Cutting Planes Exploiting Structure

In this section we follow a different route to derive cutting planes and analyze the structure of the constraint matrix. The idea is to identify some substructure of $Ax \leq b$ and use the polyhedral knowledge about this substructure to strengthen the original formulation. Let $A_{IJ} x_J \leq b_I$ with

$I \subseteq M, J \subseteq N \cup C$ be such a subsystem of (2). If $J = N \cup C$, we have that $P_{\text{IP}} \subseteq \{x \in \mathbb{Z}^N \times \mathbb{R}^C : A_I.x \leq b_I\} =: P_{\text{rel}}$ and any cutting plane valid for P_{rel} is also valid for P_{IP}. Thus the task is to identify some substructure where one knows (part of) the polyhedral structure and to find violated cutting planes for this substructure. This approach was initiated by Crowder, Johnson and Padberg [20] for 0/1 integer programs, where each row of the constraint matrix was interpreted as a knapsack problem. Since this approach is still very common to many MIP solvers and is still very successful, we describe some of the cutting planes known for the 0/1 knapsack polytope that are used to strengthen general mixed integer programs. In case J is a proper subset of $N \cup C$, a valid inequality for P_{rel} is not necessarily valid for P_{IP}. In this case we have to resort to *lifting*. The main idea of lifting will be described at the end of this section. As we will see lifting is also useful to strengthen valid inequalities.

Knapsack Relaxations

Consider the following polytope

$$P_K(N, f, F) := \text{conv}\{x \in \{0,1\}^N : \sum_{i \in N} f_j x_j \leq F\} \tag{19}$$

with some finite set N, weights $f_j \in \mathbb{Q}$, $j \in N$, and a capacity $F \in \mathbb{Q}$. $P_K(N, f, F)$ is called the 0/1 *knapsack polytope*. We obtain a knapsack relaxation from our integer program (2) by taking some row i and setting $f_j = a_{ij}$ and $F = b_i$, where we assume that all involved variables are binary. Thus any valid inequality for $P_K(N, f, F)$ is also valid for P_{IP}. In the following we summarize some of the inequalities known for the 0/1 knapsack polytope that are also used for the solution of integer programs.

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

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

that is valid for the knapsack polyhedron $P_K(N, f, F)$. If the cover is *minimal*, i. e., if $\sum_{i \in S \setminus \{s\}} f_i \leq F$ for all $s \in S$, the inequality is called *minimal cover inequality (with respect to S)*. In [4, 56, 36, 68] it was shown that the minimal cover inequality defines a facet of $P_K(S, f, F)$.

Another well-known class of knapsack inequalities are $(1, k)$-configuration inequalities that were introduced by Padberg [57]. A $(1, k)$-*configuration* consists of a feasible set S, i. e., a set S such that $\sum_{i \in S} f_i \leq F$, plus one additional item z such that every subset of S of cardinality k, together with z, forms a minimal cover. A $(1, k)$-configuration $S \cup \{z\}$ gives rise to the inequality

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

which is called a $(1, k)$-*configuration inequality (with respect to $S \cup \{z\}$)*. Note that a minimal cover S is a $(1, |S| - 1)$-configuration, and vice versa, a $(1, k)$-configuration inequality (with respect to $S \cup \{z\}$) that satisfies $k = |S|$ is a minimal cover. In [57] it was shown that the $(1, k)$-configuration inequality defines a facet of $P_K(S \cup \{z\}, f, F)$.

Inequalities derived from both covers and $(1, k)$-configurations are special cases of *extended weight inequalities* that have been introduced by Weismantel [67]. Consider a subset $T \subseteq N$ with $f(T) < F$ and let $r := F - f(T)$. The inequality

$$\sum_{i \in T} f_i x_i + \sum_{i \in N \setminus T} (f_i - r)^+ x_i \leq f(T). \tag{20}$$

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

is a natural way to extend weight inequalities by (i) replacing the original weights of the items by relative weights and (ii) resorting to the method of sequential lifting.

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

$$\sum_{i \in T} x_i + \sum_{i \in S} w_i x_i \leq |T|, \tag{21}$$

where $w_i, i \in S$, are the lifting coefficients obtained by applying Algorithm 2 on page 14. These (uniform) extended weight inequalities subsume the family of minimal cover and $(1,k)$-configuration inequalities. They can be generalized to inequalities with arbitrary weights in the starting set T, see [67].

The separation of minimal cover inequalities is widely discussed in the literature. The complexity of cover separation has been investigated in [23, 41, 32], whereas algorithmic and implementational issues are treated among others in [20, 33, 37, 61, 71]. 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), and lift the remaining variables sequentially.

Cutting planes derived from knapsack relaxations can sometimes be strengthened if special ordered set (SOS) inequalities $\sum_{i \in Q} x_i \leq 1$ 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 [20, 40, 70, 53, 33].

Lifting

As outlined at the beginning of this section and as observed in the description and separation of knapsack inequalities we are often faced with the following problem.

We are given an inequality $\sum_{i \in I} \alpha_i x_i \leq \alpha_0$ that is valid for $P_{\text{IP}} \cap \{x \in \mathbb{R}^n : x_j = 0 \text{ for all } j \in N \setminus I\}$ for some $I \subseteq N$. We would like to extend this inequality to a valid inequality of P_{IP} and, if possible, in such a way that it induces a high dimensional face of P_{IP}. Or in case $\sum_{i \in I} \alpha_i x_i \leq \alpha_0$ is already facet-defining for the subpolytope (as for instance the minimal cover for $P_K(S, f, F)$), we would like to extend the inequality to a facet-defining inequality of P_{IP} (in the minimal cover case to a facet-defining inequality of $P_K(N, f, F)$). One way to solve this problem is the *method of sequential lifting*, see [56, 68]. The algorithm proceeds in an iterative fashion. It takes into account step by step a variable $i \in N \setminus I$, computes an appropriate coefficient α_i for this variable and iterates. We assume in the following that $\pi_1, \ldots, \pi_{n-|I|}$ is a permutation of the items in $N \setminus I$.

Algorithm 2. (Sequential lifting)

(1) **For** $k = 1$ to $n - |I|$ perform the following steps:
(2) **For** $l = 1$ to u_{π_k} perform the following steps:

$$\begin{aligned} \gamma(k,l) = \max & \sum_{i \in I} \alpha_i x_i + \sum_{i \in \{\pi_1, \ldots, \pi_{k-1}\}} \alpha_i x_i \\ & \sum_{i \in I} A_{\cdot i} x_i + \sum_{i \in \{\pi_1, \ldots, \pi_{k-1}\}} A_{\cdot i} x_i + A_{\cdot \pi_k} l \leq b \\ & 0 \leq x_i \leq u_i, x_i \in \mathbb{Z} \text{ for } i \in I \cup \{\pi_1, \ldots, \pi_{k-1}\}. \end{aligned}$$

(3) **End(For)**
(4) Set

$$\alpha_{\pi_k} := \min_{l=1,\ldots,u_{\pi_k}} \frac{\alpha_0 - \gamma(k,l)}{l}.$$

(5) **End(For)**
(6) **Stop.**

It can be shown by induction on k that the output of this algorithm $\sum_{i \in N} \alpha_i x_i \leq \alpha_0$ is a valid inequality for P_{IP}. In case, for some $k \in \{\pi_1, \ldots, \pi_{n-|I|}\}$, the integer program in (2) is infeasible,

i. e., $\gamma(k,l) = -\infty$, for all $l = 1, \ldots, u_k$, we may assign any value to α_k and the inequality stays valid. In fact, the following result is true.

Proposition 1. *Let $I \subseteq N$ and $\sum_{i\in I} \alpha_i x_i \leq \alpha_0$ an inequality that defines a facet of $P_{IP} \cap \{x \in \mathbb{R}^n : x_j = 0$ for all $j \in N \setminus I\}$. After applying Algorithm 2, the inequality $\sum_{i\in N} \alpha_i x_i \leq \alpha_0$ defines a facet of P_{IP}.*

The inequality that results from applying the lifting procedure is dependent on the permutation of the items in the set $N \setminus I$.

Example 1. Consider the knapsack polyhedron

$$P_{\text{IP}} = \text{conv}\{x \in \{0,1\}^6 : 5x_1 + 5x_2 + 5x_3 + 5x_4 + 3x_5 + 8x_6 \leq 17\}.$$

The inequality $x_1 + x_2 + x_3 + x_4 \leq 3$ is valid for $P_{\text{IP}} \cap \{x_5 = x_6 = 0\}$. Choosing the permutation $(5,6)$ yields the inequality $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3$. If one chooses the permutation $(6,5)$ of the items 5 and 6, the resulting inequality reads $x_1 + x_2 + x_3 + x_4 + 2x_6 \leq 3$. Both inequalities are facet-defining for P_{IP}.

Note that in order to perform the lifting procedure one needs to solve a couple of integer programs that - at first sight - appear as difficult as the original problem. Sometimes they are not. For instance, if the integer program is a 0/1 knapsack problem and the starting inequality $\sum_{i\in I} \alpha_i x_i \leq \alpha_0$ is a minimal cover or $(1,k)$-configuration inequality, the lifting coefficients can be computed in polynomial time, see [71]. Sometimes it is possible to determine the exact lifting coefficient without solving integer programs, as was observed by Balas [4] for minimal cover inequalities and extended by Weismantel [67] to extended weight inequalities. It is however true that for many general mixed integer programs the lifting procedure can hardly be implemented in the way we presented it, because computing the coefficients step by step is just too expensive. In such cases, one resorts to lower bounds on the coefficients that one obtains from heuristics. Another way is to look for conditions under which simultaneous lifting of variables is possible. This leads to the study of superadditive functions [69, 35].

We note that lifting can, of course, also be applied if a variable x_i is currently at its upper bound u_i. In this case, we first "complement" variable x_i by replacing it by $u_i - x_i$, apply the same Algorithm 2 and resubstitute the variable afterwards. Lifting (sequential or simultaneous) has also been applied to general mixed integer programs, see, for instance, [34, 47] or in connection with lift-and-project cuts, see [7, 5] and the article in this book.

Computational results about the success of knapsack inequalities with or without GUB constraints are given, for instance, in [20, 11, 19, 33, 48]. The papers consistently show that knapsack cuts are crucial for the solution of integer programs that contain knapsack problems as a substructure.

Of course, knapsack relaxations are not the only ones considered in mixed integer programming solvers. An analysis of other important relaxations of an integer program allows to incorporate odd hole and clique inequalities for the stable set polyhedron [55] or flow cover inequalities for certain mixed integer models [60, 61]. Further recent examples of this second approach are given in [15, 47]. More than one knapsack constraint at a time are considered in [49]. Cordier et al. [19] give a nice survey on which of the mentioned cutting planes help to solve which problems from the `Miplib`. A comprehensive survey on cutting planes used to solve integer and mixed integer programs is given in [46].

6 Conclusions

In this paper we have discussed the basic features of current branch-and-cut algorithms for the solution of mixed integer programs. We have especially seen that preprocessing, though most of the ideas are straight-forward, is often very important to solve certain mixed integer programs. We have also observed that there are various alternative and better strategies for node and variable selection within the branch-and-bound enumeration scheme than the classical choices of selecting

some node deepest in the tree and selecting a variable closest to one half. We also got to know some cutting planes that are incorporated into todays software. Of course, we could just touch the surface of these topics in this survey. The interested reader is most welcome to get deeper into the field through the cited literature.

References

1. 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.
2. E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Mathematical Programming*, 71:221 – 245, 1995.
3. D. Applegate, R. E. Bixby, V. Chvátal, and W. Cook. Finding cuts in the TSP. Technical Report 95-05, DIMACS, March 1995.
4. E. Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8:146 – 164, 1975.
5. 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.
6. 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.
7. E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1 – 9, 1996.
8. 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.
9. R. E. Bixby. *Lectures on Linear Programming*. Rice University, Houston, Texas, Spring 1994.
10. R. E. Bixby, S. Ceria, C. McZeal, and M. W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. Information on the paper and the problems are available at WWW Page: http://www.caam.rice.edu/∼bixby/miplib/miplib.html, 1998.
11. 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.
12. R. Borndörfer. *Aspects of Set Packing, Partitioning, and Covering*. PhD thesis, Technische Universität Berlin, 1998.
13. R. Borndörfer, C. E. Ferreira, and A. Martin. Decomposing matrices into blocks. *SIAM Journal on Optimization*, 9:236 – 269, 1998.
14. Alberto Caprara and Matteo Fischetti. Branch-and-cut algorithms. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 45–63. John Wiley & Sons Ltd, Chichester, 1997.
15. 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.
16. V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305 – 337, 1973.
17. Vašek Chvátal. *Linear Programming*. W. H. Freeman and Company, 1983.
18. J.M. Clochard and D. Naddef. Using path inequalities in a branch-and-cut code for the symmetric traveling salesman problem. In Lawrence Wolsey and Giovanni Rinaldi, editors, *Proceedings on the Third IPCO Conference*, pages 291–311, 1993.
19. 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.
20. H. Crowder, E. Johnson, and M. W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.
21. DASH Associates, Blisworth House, Blisworth, Northants NN7 3BX, UK. *XPRESS-MP Optimisation Subroutine Library*, 2001. Information available at URL http://www.dash.co.uk.
22. J. Eckstein. Parallel branch-and-bound algorithms for general mixed integer programming on the CM-5. *SIAM Journal on Optimization*, 4:794 – 814, 1994.
23. C. E. Ferreira. *On Combinatorial Optimization Problems Arising in Computer System Design*. PhD thesis, Technische Universität Berlin, 1994.
24. C. E. Ferreira, A. Martin, and R. Weismantel. Solving multiple knapsack problems by cutting planes. *SIAM Journal on Optimization*, 6:858 – 877, 1996.
25. R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society*, 64:275 – 278, 1958.
26. R. E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The RAND Cooperation, 1960.

27. 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.
28. R. E. Gomory. An algorithm for integer solutions to linear programming. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269 – 302, New York, 1969. McGraw-Hill.
29. J. Gondzio. Presolve analysis of linear programs prior to apply an interior point method. *INFORMS Journal on Computing*, 9:73 – 91, 1997.
30. M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169 – 197, 1981.
31. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.
32. 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.
33. 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.
34. 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.
35. 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.
36. P. L. Hammer, E. .L. Johnson, and U. N. Peled. Facets of regular 0-1 polytopes. *Mathematical Programming*, 8:179 – 206, 1975.
37. 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.
38. 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.
39. 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.
40. E. Johnson and M. W. Padberg. A note on the knapsack problem with special ordered sets. *Operations Research Letters*, 1:18 – 22, 1981.
41. D. Klabjan, G. L. Nemhauser, and C. Tovey. The complexity of cover inequality separation. *Operations Research Letters*, 23:35 – 40, 1998.
42. T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207 – 232, 1998.
43. A. Land and S. Powell. Computer codes for problems of integer programming. *Annals of Discrete Mathematics*, 5:221 – 269, 1979.
44. J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. Technical Report LEC-97-12, Georgia Institute of Technology, 1997.
45. 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.
46. 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.
47. H. Marchand and L. A. Wolsey. The $0-1$ knapsack problem with a single continuous variable. *Mathematical Programming*, 85:15 – 33, 1999.
48. A. Martin. Integer programs with block structure. Habilitations-Schrift, Technische Universität Berlin, 1998.
49. 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.
50. G. Mitra. Investigations of some branch and bound strategies for the solution of mixed integer linear programs. *Mathematical Programming*, 4:155 – 170, 1973.
51. 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.
52. G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47 – 58, 1994.
53. 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.
54. 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.
55. M. W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

56. M. W. Padberg. A note on zero-one programming. *Operations Research*, 23:833–837, 1975.
57. M. W. Padberg. $(1, k)$-configurations and facets for packing problems. *Mathematical Programming*, 18:94–99, 1980.
58. M. W. Padberg. *Linear Optimization and Extensions*. Springer, 1995.
59. M. W. Padberg. Classical cuts for mixed-integer programming and branch-and-cut. Technical report, New York University, 2000. To appear in MMOR.
60. M. W. Padberg, T. J. Van Roy, and L. A. Wolsey. Valid inequalities for fixed charge problems. *Operations Research*, 33:842 – 861, 1985.
61. T. J. Van Roy and L. A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35:45 – 57, 1987.
62. A. Schrijver. On cutting planes. *Annals of Discrete Mathematics*, 9:291 – 296, 1980.
63. Ramesh Sharda. Linear programming solver software for personal computers: 1995 report. *OR/MS Today*, 22(5):49 – 57, 1995.
64. Uwe H. Suhl and R. Szymanski. Supernode processing of mixed-integer models. *Computational Optimization and Applications*, 3:317 – 331, 1994.
65. Stefan Thienel. *ABACUS A Branch-And-CUt System*. PhD thesis, Universität zu Köln, 1995.
66. J. A. Tomlin and J. S. Welsh. Finding duplicate rows in a linear program. *Operations Research Letters*, 5:7 – 11, 1986.
67. R. Weismantel. On the 0/1 knapsack polytope. *Mathematical Programming*, 77:49–68, 1997.
68. L. A. Wolsey. Faces of linear inequalities in 0-1 variables. *Mathematical Programming*, 8:165 – 178, 1975.
69. L. A. Wolsey. Valid inequalities and superadditivity for 0/1 integer programs. *Mathematics of Operations Research*, 2:66 – 77, 1977.
70. L. A. Wolsey. Valid inequalities for 0-1 knapsacks and MIPs with generalized upper bound constraints. *Discrete Applied Mathematics*, 29:251–261, 1990.
71. E. Zemel. Easily computable facets of the knapsack polytope. *Mathematics of Operations Research*, 14:760 – 764, 1989.