COMBINATORIAL OPTIMIZATION AND INTEGER PROGRAMMING

Michael Jünger, Institut für Informatik, Universität zu Köln, Germany
Gerhard Reinelt, Institut für Informatik, Universität Heidelberg, Germany

Keywords
mixed integer program, combinatorial optimzation problem, linear program, NP-hard, polyhedra, relaxation, branch-and-bound, cutting plane

Short contents list

Glossary

$\mathbf{R}$: set of real numbers
$\mathbf{R}^E$: real vector space of dimension $|E|$ where the components are indexed by the elements of E
2^E: power set of E
halfspace: $\{x \in \mathbf{R}^n \mid ax \leq a_0\}$, for some $a \neq 0$
hyperplane: $\{x \in \mathbf{R}^n \mid ax = a_0\}$, for some $a \neq 0$
conic hull of X: $\text{cone}(X) = \{x \mid x = \sum_{i=1}^m \alpha_i x^i, x^i \in X, \alpha_i \geq 0,$ for some $m > 0\}$
convex hull of X: $\text{conv}(X) = \{x \mid x = \sum_{i=1}^m \alpha_i x^i, x^i \in X, \alpha_i \geq 0, \sum_{i=1}^m \alpha_i = 1,$

for some $m > 0\}$
polyhedron: $\{x \in \mathbf{R}^n \mid Ax \leq b\}$ for $A \in \mathbf{R}^{m\times n}$ and $b \in \mathbf{R}^m$, or $\mathrm{conv}(X) + \mathrm{cone}(Y)$, for finite sets X and Y
linear program (LP): $\max\{cx \mid Ax \leq b\}$, $\max\{cx \mid Ax = b, x \geq 0\}$
integer program (IP): $\max\{cx \mid Ax \leq b, x \text{ integer}\}$
mixed integer program (MIP): $\max\{cx \mid Ax \leq b, x^i \text{ integer for } i \in I\}$, $I \subseteq \{1, 2, \dots, n\}$
0-1 integer program (0-1-IP): $\max\{cx \mid Ax \leq b, x \in \{0,1\}^n\}$
combinatorial optimization problem: $\max\{f(F) \mid F \in \mathcal{F}\}$ where $\mathcal{F} \subseteq 2^E$ for a finite set E

Summary

Solution techniques for combinatorial optimization and integer programming problems are core disciplines in operations research with contributions of mathematicians as well as computer scientists and economists. This article surveys the state of the art in solving such problems to optimality.

1 Introduction

Combinatorial optimization and integer programming is concerned with finding optimum solutions for optimization problems that involve yes/no decisions or determining optimum levels of discrete quantities. Research in solution techniques and corresponding computer software originated in the fifties and has been florishing especially in the last decade. Our overview of the current state of the art is organized as follows.
In section 2 we introduce a few illuminating example problems and generalize to the generic models that are the subject of this article. In section 3 we sketch the mathematical foundations of today's solution techniques, section 4 deals with the most important algorithmic approaches, and section 5 concludes our exposition with general remarks on the availability of these techniques as computer software.

2 Modeling

2.1 Example Applications

Many of the most well-known operations research problems can be formulated as (mixed) integer linear programs. Before generic (mixed) integer programming models as well as a generic combinatorial optimization model are introduced, a few examples are considered.
In the *assignment problem (AP)*, persons must be assigned to jobs, say, n persons to n jobs. If p_{ij} denotes the level of proficiency person i posesses for job j, the task is to find an assignment that maximizes the total proficiency. Let the unknown x_{ij} be either 1 or 0 if person i is assigned to job j or not, respectively. Then the problem can be written as

$$\begin{aligned} \max \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}x_{ij} & \\ \sum_{i=1}^{n} x_{ij} = 1, &\qquad \text{for all } j \in \{1,2,\dots,n\}, \\ \sum_{j=1}^{n} x_{ij} = 1, &\qquad \text{for all } i \in \{1,2,\dots,n\}, \\ x_{ij} \in \{0,1\}, &\qquad \text{for all } i,j \in \{1,2,\dots,n\}. \end{aligned} \tag{1}$$

The equations make sure that each person is assigned to exactly one job and each job to exactly one person.

For a more compact formulation of the assignment problem (and some of the subsequent example problems) it is convenient to introduce *undirected graphs* $G = (V, E)$ with a finite *node* set V and *edge* set $E \subseteq \{\{u, w\} \mid u, w \in V,\ u \neq w\}$. Two edges are *adjacent* if they share a common node. The complete bipartite graph $K_{n,n}$ is the graph $K_{n,n} = (V, E)$ with node set $V = U \cup W$, where $|U| = n$, $|W| = n$, $U \cap W = \emptyset$, and edge set $E = \{\{u, w\} \mid u \in U,\ w \in W\}$. Let $\mathbf{R}^E$ denote the real vector space of dimension $|E|$ where the components are indexed by the elements of E. For a set $F \subseteq E$ its *characteristic vector* $x^F \in \mathbf{R}^E$ is defined by setting $x^F_e = 1$, if $e \in F$, and $x^F_e = 0$, otherwise. An assignment then is a set of n pairwise nonadjacent edges of $K_{n,n}$ and the x_{ij} become its characteristic vector.
For a graph $G = (V, E)$ and a node set $W \subseteq V$ let $\delta(W) := \{\{u, w\} \in E \mid |\{u, w\} \cap W| = 1\}$ denote the *cut* induced by W, i.e. the edges with one endnode in W and the other in $V \setminus W$, and let $\delta(v)$ be a shorthand for $\delta(\{v\})$. For an edge set $F \subseteq E$ and variables x_e ($e \in E$) let $x(F) := \sum_{e \in F} x_e$. Furthermore, for $a, b \in \mathbf{R}^n$, let $ab := \sum_{i=1}^n a_i b_i$ denote the inner product of a and b. Then the assignment problem can equivalently be written as

$$\begin{aligned} &\max\ px \\ x(\delta(v)) = 1, &\quad \text{for all } v \in V, \\ x_e \in \{0,1\}, &\quad \text{for all } e \in E. \end{aligned} \tag{2}$$

The related *perfect matching problem (PMP)* arises if one wants to find an optimum pairing of an even number $2n$ of items where each pairing of i and j induces a profit p_{ij}. E.g., assigning students to double rooms in a dormitory is such a task where there is no bipartition like in the assignment problem. In graph theoretic terms, the problem is to determine n pairwise nonadjacent edges in a complete graph $K_{2n} = (V, E)$ where $|V| = 2n$ and $E = \{\{u, w\} \mid u, w \in V,\ u \neq w\}$, with edge weights p_{ij}. It can be modeled just like (1) with an underlying complete graph K_{2n} instead of a complete bipartite graph $K_{n,n}$.
A traveling salesman, starting in his home city, must visit each of additional $n - 1$ cities exactly once and return to his home. If d_{ij} denotes the distance between towns i and j (where $d_{ij} = d_{ji}$), one of his problems, the *traveling salesman problem (TSP)*, consists of choosing a tour of minimum distance traveled.
The problem can be modeled on a complete graph with edges of corresponding to the direct connections between two cities weighted according to the distances. Variables $x_{ij} \in \{0, 1\}$ are introduced with the interpretation that $x_{ij} = 1$ if the salesman uses the edge between i and j and $x_{ij} = 0$ otherwise. Using the notation $E(W) = \{\{i, j\} \mid i \in W, j \in W\}$ the task can then be formulated as

$$\begin{aligned} &\min\ dx \\ x(\delta(v)) = 2, &\quad \text{for all } v \in V, \\ x(E(W)) \leq |W| - 1, &\quad \text{for all } W \subset V,\ 3 \leq |W| \leq n - 2, \\ x_e \in \{0,1\}, &\quad \text{for all } e \in E. \end{aligned} \tag{3}$$

The equations, called the *degree constraints*, express that in a tour, each city is touched by two direct connections, i.e. each node of the graph is incident with exactly two tour edges. The inequalities, called *connectivity constraints*, make sure that each nonempty subset of cities other than the whole set V is entered and left, thus excluding short cycles.
Let $N = \{1, 2, \ldots, n\}$ be a set of potential fire station locations and $M = \{1, 2, \ldots, m\}$ a set of communities to be protected. For $j \in N$ let $M_j \subseteq M$ denote the set of communities that can be reached from location j in less than 10 minutes, and c_j denote the cost of building a fire station at location j. The task to decide which stations to build at least possible cost such that all communities are protected can be formulated as a *set covering problem (SCP)*.

Let $B = (b_{ij}) \in \mathbf{R}^{m\times n}$ be the matrix whose columns are the characteristic vectors of the sets M_j, $j = 1, 2, \ldots, n$, i.e., $b_{ij} \in \{0, 1\}$ and $b_{ij} = 1$ if and only if community i can be protected by station j. If **1** denotes the vector of all 1's, then one has to solve the following problem.

$$\begin{aligned} \min\ & cx \\ Bx &\geq \mathbf{1} \\ x &\in \{0,1\}^n. \end{aligned} \tag{4}$$

This type of problem belongs to the broad class of location problems.
For the classical *facility location problem (FLP)*, the input consists again of a set $N = \{1, 2, \ldots, n\}$ of potential facilities and a set $M = \{1, 2, \ldots, m\}$ of clients with demands of a certain good supplied from the facilities. Facility $j \in N$, if built at cost c_j, has a capacity of u_j, the demand of client $i \in M$ is b_i, and it costs h_{ij} to satisfy a unit of i's demand from facility j. If $x_j \in \{0, 1\}$ encodes the decision of whether or not facility j is opened and the continuous variable y_{ij} the quantity of i's demand that is satisfied from facility j, then the problem is the following.

$$\begin{aligned} \min \sum_{j\in N} c_j x_j + \sum_{i\in M}\sum_{j\in N} h_{ij} y_{ij} & \\ \sum_{j\in N} y_{ij} = b_i, &\quad \text{for all } i \in M, \\ \sum_{i\in M} y_{ij} - u_j x_j \leq 0, &\quad \text{for all } j \in N, \\ y_{ij} \geq 0, &\quad \text{for all } i \in M,\ j \in N, \\ x_j \in \{0,1\}, &\quad \text{for all } j \in N. \end{aligned} \tag{5}$$

The equations guarantee that all demands are satisfied and the inequalities ensure that only opened facilities are used and their capacities are not exceeded.
A picture frame consists of two horizontal and two vertical parts. A picture frame manufacturer needs to cut m frame parts of lengths $a_1, a_2, \ldots, a_m$ from base rods of length L. For producing the parts he wishes to determine a cutting strategy that minimizes the number of used base rods. Defining M and N as above, one can use analogous techniques as in (5) to formulate the problem. Let $n \leq m$ be an upper bound on the number of required base rods, then the so-called *cutting stock problem (CSP)* is the following.

$$\begin{aligned} \min \sum_{j\in N} x_j & \\ \sum_{j\in N} y_{ij} = 1, &\quad \text{for all } i \in M, \\ \sum_{i\in M} a_i y_{ij} - L x_j \leq 0, &\quad \text{for all } j \in N, \\ y_{ij} \in \{0,1\}, &\quad \text{for all } i \in M,\ j \in N, \\ x_j \in \{0,1\}, &\quad \text{for all } j \in N. \end{aligned} \tag{6}$$

In any feasible solution $x_j = 1$ if and only if base rod j is used and $y_{ij} = 1$ if and only if part i is cut from base rod j.
Two further basic problems are of interest in combinatorial optimization. Like the examples above, they can be formulated in terms of maximizing or minimizing a linear objective function subject to linear constraints and integrality conditions, yet such a description is of less interest here.
A *(v_1, v_k)-path* in $G = (V, E)$ is a set of edges $\{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}\}$ with distinct nodes $v_1, v_2, \ldots, v_k \in V$. A graph is *connected* if either $|V| = 1$ or for each pair of distinct nodes $u, w \in V$ there exists a (u, w)-path in G. A *cycle* arises if all nodes are distinct except $v_1 = v_k$.

Both problems are formulated for an undirected graph G with edge weights. In the *minimum spanning tree problem (MSTP)*, the objective is to find a *spanning tree*, i.e., a connected subgraph with no cycles (i.e., with $n-1$ edges) of minimum total weight. In the (u,w)*-shortest path problem (ShPP)*, two vertices $u, w \in V$ are given and the objective is to determine a (u,w)-path in G of minimum total weight.

2.2 Generic Models

A *linear mixed integer optimization problem (MIP)* is defined by a matrix $A \in \mathbf{R}^{m\times n}$, vectors $b \in \mathbf{R}^m$, $c \in \mathbf{R}^n$, and a subset $I \subseteq \{1,2,\dots,n\}$, and is formulated as follows.

$$\begin{aligned} &\max\ cx \\ &Ax \le b \\ &x_i \text{ integer}, \quad \text{for all } i \in I. \end{aligned} \tag{7}$$

The task is either to find an optimum solution of this problem or to prove that no solution exists or that the objective function is unbounded over the feasible set. If $I = \emptyset$, the problem is a *linear optimization problem (LP)*, and if $I = \{1,2,\dots,n\}$ the problem becomes the *linear integer optimization problem (IP)*. Special cases of the MIP arise if "integer" is replaced by $\{0,1\}$, and are called *0-1-MIP* and *0-1-IP*, respectively. Notice that it makes no difference if the objective function is to be minimized or maximized, since by replacing c by $-c$ one problem type can be converted to the other. All problem formulations in the previous subsection as well as many others are examples of these generic models. The set covering model (4) and the cutting stock model (6) are 0-1-IP's, while the facility location problem (5) is a 0-1-MIP.
The assignment problem, the perfect matching problem, and the traveling salesman problem (as well as the minimum spanning tree problem and the (u,w)-shortest path problem) are examples of the following generic *combinatorial optimization problem (COP)*. Given a collection $\mathcal{F} \subseteq 2^E$ on some finite ground set E and an objective function $f : \mathcal{F} \longrightarrow \mathbf{R}$, the task is to determine a subset $F^* \in \mathcal{F}$ that maximizes (or minimizes) f on $\mathcal{F}$, i.e., $f(F^*) \ge f(F)$ (or $f(F^*) \le f(F)$) for all $F \in \mathcal{F}$. In the above example problems, E is the edge set of some undirected graph and $\mathcal{F}$ consists of those edge subsets that induce subgraphs that are assignments, matchings, and tours, respectively. The example problems also exhibit a special feature which they share with many other combinatorial optimization problems, namely, f is defined on the elements of the ground set E and is extended to 2^E via the linear function $f(F) = \sum_{e\in F} f(e)$. A combinatorial optimization problem is characterized by the triple $(E, \mathcal{F}, f)$.

3 Mathematical Foundations

A particular feature of combinatorial and (mixed) integer programing problems is their "discrete nature" caused by the fact that all or many of the variables are only allowed to take integer values. Therefore the set of feasible solutions is not connected, but consists of separate sets or possibly of discrete points only. The main consequence is that the powerful methods from continuous optimization cannot be applied. (As will become clear later they can, however, be utilized to some extent.) This section reviews basic mathematical facts necessary for developing effective algorithms for solving combinatorial and integer problems to (proven) optimality.

3.1 Complexity

Complexity theory is concerned with the difficulty of problems, i.e. with the question of how fast a solution algorithm for a problem can be. Here an algorithm is assessed by giving the asymptotic

increase of its worst-case running time depending on the size of the given problem instance. An algorithm is said to have *(worst-case) running time* $O(p)$ with a function p if its running time is at most $p(s)$ for any problem of size s, where the size of a problem instance can more or less be taken as the size of its representation on a digital computer. For the minimum spanning tree problem one can easily find solution algorithms with running time $O(n^2)$, i.e., where the execution time grows only quadratically with the number of nodes. For the traveling salesman problem, however, nobody succeeded so far in developing an algorithm whose running time is less than exponential in the number of nodes. This substantial difference between the difficulty of problems has been exactly defined using methods from computational complexity theory. For the purposes of this article it is sufficient to know that there is the rigorously defined concept of *NP-hard optimization problems* which have an inherent difficulty that makes it very likely that for such problems no algorithms with polynomial running times exist. Though the proof or disproof of this conjecture belongs to the great unsolved problems in computer science, it is a well-accepted working hypothesis that there is no polynomial algorithm for solving an NP-hard optimization problem to optimality. For a thorough discussion of the complexity of algorithms see: Complexitiy Theory.
The general integer and mixed-integer problem is NP-hard and so are most of the combinatorial optimization problems that are practically relevant. Interestingly, already the *knapsack problem* $\max\{cx \mid ax \le b, x \in \{0,1\}\}$, i.e. the optimization of linear function over one inequality constraint in binary variables, is NP-hard.
This article deals with techniques that can effectively be applied to solving NP-hard problems in practice. These methods are not polynomial, but have been used with great success.
It should be noticed that it is not the number of feasible solutions that makes a problem difficult. For a minimum spanning tree problem on n nodes there are n^{n-2} possible solutions (in a complete graph) and for the assignment problem for n persons and n jobs there are $n!$ feasible solutions, but both problems are easily solved in polynomial time.

3.2 Polyhedra

Polyhedra are of central importance in the following, and some concepts have to be reviewed (for a complete discussion see: Polyhedral Theory).
For $\alpha \in \mathbf{R}^n$, $\alpha \ne 0$, and $\alpha_0 \in \mathbf{R}$ the set $\{x \in \mathbf{R}^n \mid \alpha x = \alpha_0\}$ is called *hyperplane* with associated *halfspace* $\{x \in \mathbf{R}^n \mid \alpha x \le \alpha_0\}$. A *polyhedron* P is defined as the intersection of finitely many halfspaces or equivalently as the solution set of a finite system of linear inequalities. More precisely, P is a polyhedron if there exists a matrix $A \in \mathbf{R}^{m\times n}$ and a vector $b \in \mathbf{R}^m$ such that $P = \{x \in \mathbf{R}^n \mid Ax \le b\}$. Such a description of a polyhedron by means of linear inequalities is called *outer description*. Notice that some inequalities may actually be equations. By describing P as $P = \{x \in \mathbf{R}^n \mid Ax \le b, Bx = d\}$ it is emphasized that equations are present.
For $X = \{x^1, x^2, \dots, x^n\}$ and $\alpha_i \in \mathbf{R}$, $i \in \{1, 2, \dots, n\}$, the linear combination $x = \sum_{i=1}^n \alpha_i x^i$ is called a *conic combination* of the elements of X if $\alpha_i \ge 0$ for all $i \in \{1, 2, \dots, n\}$ and, if in addition $\sum_{i=1}^n \alpha_i = 1$, the vector x is called a *convex combination* of the elements of X. We denote the set of all conic and convex combinations of the elements of X by $\mathrm{cone}(X)$ and $\mathrm{conv}(X)$, respectively.
According to classical results in polyhedral theory it is possible to describe polyhedra in an alternative way. Namely, P is a polyhedron if and only if there exist finite sets X and Y such that P is the sum of the convex hull of X and the conic hull of Y, i.e., $P = \{z \mid z = x + y \text{ for some } x \in \mathrm{conv}(X) \text{ and } y \in \mathrm{cone}(Y)\}$. This type of description is called *inner description*.
A *polytope* is a bounded polyhedron, i.e., P is a polytope if and only if it is a polyhedron and if there exist bounds l, u such that $l \le x \le u$ for all $x \in P$. In particular, P is a polytope if and only if it is equal to the convex hull of a finite set.

The *dimension* $\dim P$ of a polyhedron $P \subseteq \mathbf{R}^n$ can be computed as follows. Let $Bx = d$ be a system of equations such that every equation that is satisfied by all points in P is either contained in this system or is a linear combination of equations of $Bx = d$. Then $\dim P = n - \operatorname{rank} B$, where $\operatorname{rank} B$ is the linear rank of the matrix B. A polyhedron $P \subseteq \mathbf{R}^n$ is said to be *full dimensional* if $\dim P = n$. Therefore, if P is full dimensional, then there does not exist an equation $ax = a_0$, $a \neq 0$ with $P \subseteq \{x \mid ax = a_0\}$.
An inequality $ax \leq a_0$, $a \neq 0$, is said to be *valid* for a polyhedron P if $P \subseteq \{x \mid ax \leq a_0\}$. If $ax \leq a_0$ is a valid inequality for P then the set $F = P \cap \{x \mid ax = a_0\}$ is a *face* of P (which may be the empty face). If $F \neq P$ then F is called a *proper face*. Each face is itself a polyhedron. If $|F| = 1$ then the element $v \in F$ is called a *vertex* of P. Vertices have the property that they cannot be represented as convex combinations of other elements of the polyhedron. If P is a polytope, then $P = \operatorname{conv}(V)$ where V is the set of vertices of P. If F is a maximal nonempty proper face then F is called a *facet* of P. A face F is a facet of P if and only if $\dim F = \dim P - 1$. If F is a facet defined by $ax \leq a_0$ then this inequality is called *facet defining* for P.
For a given polyhedron P, one is interested in its *minimal linear description* $P = \{x \mid Ax \leq b, Bx = d\}$. In such a description no equation from $Bx = d$ is implied by other equations of this system and the inequality system $Ax \leq b$ contains exactly one defining inequality for every facet of P. This fact substantiates the interest in finding facet-defining inequalities for polyhedra. They are the strongest ones since they are necessary and cannot be replaced by other inequalities.

3.3 Linear Programming

Linear programming will turn out to be a basic tool in discrete optimization. It deals with the maximization (or minimization) of a linear function over a polytope given by an outer description. Linear programming problems are usually given in the form $\max\{cx \mid Ax \leq b\}$ or $\max\{cx \mid Ax = b, x \geq 0\}$. Both formulations are equivalent, for illustrating certain facts one formulation can be preferrable to the other.
A linear program can be considered an easy problem. From a theoretical point of view there are polynomial time algorithms (ellipsoid method, interior point methods) for its solution. A subtle difference between ellipsoid and interior point methods has to be emphasized here. Interior point methods can solve the problem in time polynomial in the size of the constraint system. The running time of the ellipsoid method, however, does not depend on the number of constraints. It is already polynomial if one can check in polynomial time if a given x satisfies all constraints and produce a violated inequality in the case that it does not. This allows the ellipsoid method to optimize (in a certain sense) linear programs with an exponential number of constraints. For a complete treatment of the subject and in particular of the complexity issues given here very imprecisely see Linear Programming.
But also from a practical point of view there are effective algorithms (simplex algorithms, interior point algorithms) which are able to solve very large problem instances with several ten-thousands of constraints and millions of variables.
Let the linear program be given as $\max\{cx \mid Ax = b, x \geq 0\}$. The vertices of the underlying polyhedron are associated with so-called *bases* of A. Let x be a feasible solution, i.e. a solution that satisfies $Ax = b$, $x \geq 0$. Then x is a vertex if and only if x can be partitioned into $x = (x_B, x_N)$ such that the submatrix A_B of A corresponding to the indices in B has full row rank, $x_B = A_B^{-1}b$ and $x_N = 0$.
If the linear program is not infeasible or unbounded, then there is an optimum vertex solution and, due to duality theory, there is a short proof of optimality which can be easily checked. Namely, let x^* be a feasible vertex with associated basis B. If the *reduced costs* $c_N - c_B A_B^{-1} A_N$ are nonpositive

then x^* maximizes the objective function cx over $\{x \mid Ax = b, x \geq 0\}$. It is possible that several bases are associated with the same vertex, so although x^* might be optimal, for the chosen basis the reduced costs could have positive entries. Such degeneracy problems will be ignored throughout this article. But they have to be addressed in practical computations.

3.4 Relations Between Problems

There is a close connection between linear, (mixed) integer and combinatorial optimization problems. On the one hand, an integer programming problem can be considered as a combinatorial problem where the underlying combinatorial properties are specified by linear equations and inequalities. On the other hand, as has been done in section 2, for a combinatorial problem it is usually easy to find an integer programming formulation.

Of interest for practical computations is a link to linear programming that can be established as follows. With a combinatorial optimization problem $(E, \mathcal{F}, f)$ one can associate the polytope $P_{\mathcal{F}} = \text{conv}\{x^F \mid F \in \mathcal{F}\}$, where x^F denotes the characteristic vector of $F \in \mathcal{F}$. $P_{\mathcal{F}}$ is the convex hull of the characteristic vectors of feasible sets. Because the characteristic vectors are 0-1-vectors, they are exactly the vertices of the polytope $P_{\mathcal{F}}$, and the combinatorial optimization problem can be solved as the linear optimization problem $\max\{x \mid x \in P_{\mathcal{F}}\}$. However, to this end we need an outer description $P_{\mathcal{F}} = \{x \mid Ax \leq b\}$.

For solving a general IP or MIP $\max\{cx \mid Ax \leq b,\, x_i$ integer for all $i \in I\}$ as a linear program the availability of the linear description of the associated polyhedron $\text{conv}\{x \mid Ax \leq b,\, x_i$ integer for all $i \in I\}$ is required.

The difference between linear and integer programming formulations is usually substantial. Consider for example the traveling salesman problem on 10 cities. The integer programming formulation with subtour elimination constraints consists of 10 equations and 582 inequalities. The formulation of an equivalent linear program needs 10 equations and 51 043 900 866 inequalities, and none of them is superfluous!

Therefore it is not a general valid approach to generate a linear programming formulation for a given integer or combinatorial optimization problem. There are finite algorithms (e.g., Fourier-Motzkin-algorithm, see: Computational Linear Algebra) to convert an inner description to an outer description, but in general the outer description is simply too large to be listed explicitly and the running time of the algorithm is exponential.

On the other hand, there are IPs that are easily solved. If one replaces the constraints $x_{ij} \in \{0, 1\}$ in the IP formulation of the assignment problem by $x_{ij} \geq 0$ then one obtains immediately an LP formulation because all vertices of the associated polytope are 0-1 vertices and correspond exactly to the possible assignments. This is, however, not true for the perfect matching problem. The canonical LP relaxation defines a polytope which has non-integral vertices. An exponential number of additional inequalities $x(E(W)) \leq |W| - 1$ for sets $W \subset V$ with odd cardinality is needed. They are called the *matching inequalities*.

Given a class of inequalities $\mathcal{I}$, the *separation problem* for $\mathcal{I}$ consists of deciding whether a given $z \in \mathbf{R}^n$ satisfies all inequalities in $\mathcal{I}$ and if not, find at least one inequality $ax \leq a_0$ in $\mathcal{I}$ such that $az > a_0$, i.e. that is violated by z. In our context a (more general) theorem holds which states that an optimization problem can be solved in polynomial time if and only if the separation problem for its constraint defining inequalities can be solved in polynomial time.

Since the separation problem for the matching inequalities can be solved in polynomial time, one can also solve the perfect matching problem in polynomial time. The appropriate algorithm, however, is a combinatorial one.

3.5 Relaxations

A *relaxation* of a problem is obtained if some of the defining constraints are omitted. This way the set of feasible solutions is augmented. If the objective function is now minimized (maximized) over this enlarged set, then its optimum value gives a lower (upper) bound on the optimum objective function value for the true problem. Such bounds are useful for developing algorithmic approaches and also for giving quality guarantees for a given feasible solution. It is clear that two properties of relaxations are desired: optimization over the augmented set of solutions should be easy, but on the other hand the augmented set should be "close" to the original feasible set in order to obtain good bounds.
There are three basic types of relaxations.

3.5.1 Combinatorial Relaxation

A combinatorial optimization problem $(E', \mathcal{F}', f')$ is a relaxation of the problem $(E, \mathcal{F}, f)$, if there is an injective function $\varphi : E \to E'$ such that $\varphi(F) \in \mathcal{F}'$ and $f(F) = f'(\varphi(F))$ for all $F \in \mathcal{F}$. In most cases $E \subseteq E'$ and φ is just the identity.
Consider for example the traveling salesman problem on n nodes. It is defined by two types of constraints: the degree constraints requiring that every node is incident to exactly two edges and the connectivity constraints requiring that the set of selected edges is connected.
If the connectivity constraints are dropped, then one obtains the *2-Matching Relaxation*. The set of feasible solutions now not only contains tours but also collections of short tours. Optimum 2-matchings can be computed in time polynomial in the number of nodes.
The elimination of all degree constraints except for one node, say node 1, defines the *1-Tree Relaxation*. A feasible solution now consists of a spanning tree on the nodes $\{2, 3, \ldots, n\}$ plus two edges incident to node 1. An optimal 1-tree can be computed in polynomial time.
There are usually several possibilities to come up with combinatorial relaxations. The usefulness of different relaxations can sometimes be compared theoretically, stating that one relaxation is always tighter than another one, but in general it depends on the problem instance.

3.5.2 Linear Programming Relaxation

Linear programming relaxations for integer programming problems are obtained in a canonical way if the integrality requirement is dropped for the integer variables and just replaced by bound constraints. These relaxations are of particular interest since they can be optimized with standard algorithmic tools like simplex algorithms or interior point methods.
The canonical LP relaxation of an integer program is widely used. Its strength depends on the inequalities of the IP formulation. The choice of an inappropriate IP model can have very negative effects on the solvability of a problem. Consider as an example the facility location problem (5) where $b_i = 1$ for all $i \in M$ and $u_j = m$ for all $j \in N$. If one builds an algorithm on the canonical LP relaxation, only very small problem instances will be solved. It is advisable to replace the constraints $\sum_{i \in M} y_{ij} - m x_j \leq 0, j \in N$, by the disaggregated system

$$y_{ij} - x_j \leq 0, \quad i \in M, \;\; j \in N. \tag{8}$$

From an IP point of view both models are equivalent because they both formulate the problem correctly, but the canonical LP relaxation of the latter model is much better. This is due to the fact that the inequalities $y_{ij} - x_i \leq 0$ define facets of the convex hull of the feasible solution whereas the aggregated inequalities $\sum_{i=1}^{n} y_{ij} - n x_i \leq 0$ only define faces of lower dimension and hence lead to a weaker relaxation. Similar observations apply to the general FLP.

3.5.3 Lagrangean Relaxation

A further approach for obtaining bounds on the optimal objective function value of an integer linear program is the method of *Lagrangean relaxation*. Let an integer linear programming problem (P) be given as $\min\{cx \mid Ax \leq b, Bx = d, x \geq 0, x \text{ integer}\}$.
For any λ the integer linear programming problem

$$\begin{aligned} L(\lambda) := \min\ & cx + (Bx - d)\lambda \\ & Ax \leq b \\ & x \geq 0, \quad x \text{ integer} \end{aligned} \tag{9}$$

provides a lower bound for the optimum of (P) since every feasible solution for (P) is also feasible for the new problem with the same objective function value. The vector λ is called vector of *Lagrangean multipliers*. The best such lower bound is then given by solving the so-called *Lagrangean dual problem* $\max\{L(\lambda) \mid \lambda\}$, i.e., by finding the (unconstrained) maximum of the function L.
The function L is piecewise linear and concave (and hence nondifferentiable). Suitable methods for maximizing L are subgradient or bundle algorithms (for details see Nondifferentiable Optimization).
Clearly, the quality of the bound provided by the Lagrangean dual depends on the choice of the constraint set $Bx = d$ to be relaxed. The bound is usually better if less constraints are relaxed. On the other hand the resulting new integer program has to be easy since it must be solved many times when maximizing L. It is directly seen that also inequalities can be relaxed taking into account that the corresponding Lagrangean multipliers have to be restricted in sign.
The combinatorial 1-tree relaxation given above can be improved using Lagrangean relaxation. The following is the IP formulation of the TSP based on subtour elimination constraints where the degree constraints are partitioned into two parts and where one redundant equation, stating that exactly n edges are to be selected, is added.

$$\begin{aligned} \min\ & cx \\ x(\delta(v)) &= 2, \quad \text{for all } v \in V \setminus \{1\}, \\ x(\delta(1)) &= 2 \\ x(E) &= n \\ x(E(W)) &\leq |W| - 1, \quad \text{for all } W \subseteq V, 3 \leq |W| \leq n-2 \\ x_e &\in \{0,1\}, \quad \text{for all } e \in E. \end{aligned} \tag{10}$$

If the degree constraints for all nodes except for node 1 are relaxed, then the following Lagrange problem is obtained in which $Bx = \mathbf{2}$ denotes the system of degree constraints for the nodes $2, 3, \ldots, n$.

$$\begin{aligned} \min\ & cx + (Bx - \mathbf{2})\lambda \\ x(\delta(1)) &= 2 \\ x(E) &= n \\ x(E(W)) &\leq |W| - 1, \quad \text{for all } W \subseteq V, 3 \leq |W| \leq n-2 \\ x_e &\in \{0,1\}, \quad \text{for all } e \in E. \end{aligned} \tag{11}$$

The feasible solutions of this IP are exactly the 1-trees (which is guaranteed by the previously redundant equation). For given λ, the problem can be solved easily by a combinatorial algorithm. The simple combinatorial 1-tree relaxation corresponds to setting $\lambda = 0$. Using Lagrangean relaxation, much better 1-tree bounds can be computed. Notice that by eliminating the degree constraint for node 1 and by replacing $x_e \in \{0,1\}$ by $0 \leq x_e \leq 1$ for all $e \in E$ one obtains an LP formulation

of the minimum spanning tree problem. It has a number of constraints exponential in n, but can be solved in polynomial time (in n) because the separation problem is solvable in polynomial time.
The Lagrange bound is at least as good as the bound obtained from the canonical LP relaxation. But usually it is harder to compute it.

4 Algorithmic Approaches

Since there is a wide range of applications of integer and combinatorial optimization in practice, solving these problems is of prime concern. This article concentrates on the exact solution of problems to optimality. It is another interesting branch of research to study approximation algorithms which try to find feasible solutions close to the optimum. Questions that arise here are the development of algorithms where a quality guarantee to be achieved can be specified in advance, and the comparison of problems with respect to approximability. A discussion of these topics can be found in Approximation Algorithms.

4.1 Modeling Issues

When designing an algorithm for the solution of a problem, a very important aspect is to choose the appropriate mathematical model for its formulation. Firstly this affects the set of methods than can be used, but secondly it can be crucial for its solvability in practice.
The formulation of the cutting stock problem (6) of section 2 for example is not a suitable one. It could be strengthened using disaggregation as described above, but still the canonical LP relaxation will not be tight enough to be able so solve the problem with branch-and-bound type algorithms (see below).
Surprisingly, a seemingly much inferior description has turned out to be much more useful in practical computation. The vector $b \in \{0,1\}^m$ represents a cutting pattern for a base roll if $\sum_{i=1}^m a_i b_i \leq L$. Let the columns of the matrix $B \in \{0,1\}^{m\times n}$ represent all possible cutting patterns. Then the problem could also be modeled as the 0-1-IP

$$\begin{aligned} &\min\ \mathbf{1}z \\ &Bz = \mathbf{1} \\ z \in\ &\{0,1\}^n. \end{aligned} \tag{12}$$

In any feasible solution $z_j = 1$ if and only if the j-th cutting pattern is used.
Whereas in the previous model (with or without disaggregation) there was only a polynomial number of variables and constraints, both of order $\mathrm{O}(n^2)$, now there is a number of variables that is exponential in m. However, this problem can be dealt with as will be outlined in section 4.5. Notice the similarity of the new CSP formulation (12) with the SCP formulation (4). Because the CSP has now been modeled using equations instead of inequalites one speaks of a *set partitioning problem (SPP)*.

4.2 Polynomial Algorithms

Although the major part of practically relevant problems is NP-hard, there are several interesting combinatorial problems that are computationally easy.
Sometimes, although integer problems, they can be formulated as linear programming problems that can be solved in polynomial time. This is the case, for example, for the assignment problem as shown above. Remember that there always exists an LP formulation, but not necessarily one that is polynomially solvable in the size of the original problem. Once a suitable LP formulation is known, special variants of LP algorithms can be developed.

On the other hand there are some polynomial algorithms that do not rely on LP theory but work directly with the combinatorial structure.
Important combinatorial problems that are solvable in polynomial time are: spanning tree and arborescence problems, shortest path problems, matching and b-matching problems, transportation and network flow problems (see: Graph and Network Optimization).

4.3 Branch-and-Bound

There is a principle approach to attack a hard problem, namely, if the problem cannot be solved directly, it is split into (hopefully easier) subproblems. These subproblems are then either solved or split further until eventually very simple problems are generated. From the solution of all the generated subproblems a solution of the original problem can then be constructed. This principle is known as *divide-and-conquer*. It can be realized in various ways, in the context of discrete optimization and with the tools developed so far, the so-called *branch-and-bound* approach is most suited. Branch-and-bound (for a minimization problem) can be outlined as follows.

Branch-and-Bound Algorithm

1. Initialize the list of active subproblems with the original problem.
2. If the list of active subproblems is empty, STOP (the best feasible solution found so far is optimal).
3. Otherwise, choose some subproblem from the list of active problems and “solve” it as follows:
   (a) find an optimal solution for the subproblem, or
   (b) prove that the subproblem has no feasible solution, or
   (c) prove that there is no feasible solution for the subproblem that has smaller objective value than the best feasible solution that is already known, or
   (d) split the subproblem into further subproblems and add them to the list of active problems, if none of the above is possible.
4. Go to step (2).

The splitting of problems into subproblems can be represented by the so-called *branch-and-bound tree*, the root of which represents the original problem.
It is crucial for the efficiency of a branch-and-bound algorithm that the branch-and-bound tree does not grow too large. Therefore subproblems have to be solved if possible by alternatives (a), (b) or (c) of step 3. Alternative (a) rarely occurs, for (b) and (c) relaxations are important. Namely for (b), if a relaxation of the subproblem is already infeasible, then also the subproblem itself is infeasible. To be able to finish the subproblem in (c), good lower and upper bounds must be available. Upper bounds are obtained by finding feasible solutions. These are either obtained by solving some subproblem to optimality or usually by determining good feasible solutions using heuristics. Lower bounds can be computed by using relaxations where in principle any type of relaxation discussed above can be employed. It is clear that the tighter a relaxation the better the performance of the algorithm will be. Without a suitable relaxation branch-and-bound would tend to completely enumerate the set of feasible solutions and thus become infeasible.
The second component of this approach is *branching* which denotes the splitting of the current subproblem into a collection of new subproblems whose union of feasible solutions contains all feasible

solutions of the current subproblem. For 0-1-problems, the simplest branching rule consists of defining two new subproblems in one of which a chosen variable is required to have the value 1 in every feasible solution and in the other one to have the value 0. Other branching strategies are possible. There are also several heuristics for choosing the next subproblem to be considered.

4.4 Cutting Plane Algorithms

Most of the successful branch-and-bound algorithms for solving discrete optimization problems employ LP relaxations. The difficulties that have to be overcome when using LP relaxations will be addressed in this section.

The first problem is of general nature, namely how to solve an LP with a very large constraint set that cannot be listed explicitly. The solution is to start with a small subset of the constraint set, solve the LP to optimality, and check if the optimum solution satisfies the constraints that have not been taken into account. If this is the case, then the computed solution is optimal for the complete problem. Otherwise, there are constraints that are violated. One or some of them are added to the current LP, the LP is reoptimized, and the process continues until all constraints are satisfied. So the core problem that has to be solved here is the separation problem.

This approach is termed *cutting plane approach* due to the fact that the constraints added to the current LP "cut off" the current solution because it is infeasible for the original problem. It is important that the approach does not require an explicit list of the constraints defining the original problem. Required is "only" a method for identifying inequalities that are violated by the current solution.

The second main problem is to find strong relaxations approximating the polytope $P_{\mathcal{F}}$ defined as the convex hull of feasible solutions. In the case of combinatorial problems one can often find classes of facet-defining inequalities. For general problems, not many results about the facial structure are available, but there are methods for generating cutting planes.

A lot of effort for finding linear descriptions has been made for the traveling salesman polytope P_{TSP}. The degree equations yield a minimal equation system for P_{TSP} and therefore its dimension is $n(n-1)/2 - n$ for complete graphs. The connectivity inequalities are facet-defining (with polynomially solvable separation problem) and there are many more classes. One of them is the class of *comb inequalities*. Let H, $T_1, \ldots, T_s$ be subsets of the node set with $s \geq 3$ odd, $|T_i \cap H| \geq 1$ for $k \in \{1, \ldots, s\}$, $|T_i \setminus H| \geq 1$ for $k \in \{1, \ldots, s\}$ and $T_i \cap T_j = \emptyset$ for $i \neq j$. Then the comb inequality

$$x(E(H)) + \sum_{j=1}^{s} x(E(T_j)) \leq |H| + \frac{s-1}{2} \tag{13}$$

defines a facet of P_{TSP}. Of special interest among these inequalities are the *2-matching inequalities* which arise if all sets T_i have cardinality 2. Their corresponding separation problem can be solved in polynomial time. Based on extensive investigations very powerful TSP codes using LP relaxations and cutting planes have been implemented.

An example of general cuts for integer programs are *Gomory cuts*. Consider the integer program $\max\{cx \mid Ax = b, x \geq 0, x \text{ integer}\}$. Let A_B be an optimal basis of the canonical LP relaxation, $\overline{A} = A_B^{-1} A_N$ and $\overline{b} = A_B^{-1} b$. If $\overline{b_i}$ (the current value of the i-th basic variable) is not integral, then the inequality

$$\sum_{j \in N} (\lfloor \overline{a}_{ij} \rfloor - \overline{a}_{ij}) x_j \leq \lfloor \overline{b}_i \rfloor - \overline{b}_i \tag{14}$$

is valid for $P_{\mathcal{F}}$, but is violated by the current LP optimum. Therefore this inequality, which is easily derived, can serve as a solution of the separation problem and cuts off the current infeasible solution.

Cuts can also be derived from integral objective functions. Let $\overline{c} = c_B\overline{A} - c_N$. If c_Bx_B (the current objective function value) is not integral, then

$$\sum_{j \in N} (\lfloor \overline{c}_j \rfloor - \overline{c}_j)x_j \leq \lfloor c_Bx_B \rfloor - c_Bx_B \tag{15}$$

is a cutting plane. Cuts are added to the LP which is then reoptimized, and the process continues. Surprisingly it can be shown that this procedure (with some technical enhancements) yields an optimum integer solution in finitely many steps. There are also more general Gomory cuts and, in particular, Gomory cuts for mixed integer problems as well. In cannot be expected to solve an integer program with Gomory cuts alone, but they are very valuable for tightening relaxations and indispensable in state-of-the-art software.

Knapsack cuts arise from the observation that every single row of an IP corresponds to a knapsack problem. And inequalities defining faces or facets of the associated knapsack polytopes are at least valid for the IP. So research has been invested in finding linear descriptions of knapsack polytopes or intersections of them, and the resulting classes of inequalities are employed as cuts when solving general IPs. There are further general cuts for integer programs like for example *disjunctive cuts* or *flow cover cuts*.

At present, branch-and-bound algorithms based on LP relaxations that are solved with cutting planes are the approach of choice for many combinatorial optimization problems as well as for general (mixed) integer problems. The name *branch-and-cut* has been coined for this type of approach to distinguish it from other branch-and-bound methods.

4.5 Column Generation

Section 4.1 introduced a set partitioning model for the cutting stock problem (12) with an exponential number of variables, i.e., the constraint matrix has an exponential number of columns. If the problem is only considered on a small subset of the columns, it has to be made sure that the variables associated with the missing columns can be safely assumed to have value 0. The simplex algorithm does not only give a basic feasible solution, but also the reduced cost certificate for optimality. In this case one obtains a quantity y_i for each row i such that $\sum_{i=1}^{m} b_{ij}y_i \leq 1$ for all cutting patterns $B_{\cdot j} = (b_{1j}, b_{2j}, \ldots, b_{mj})$ that are present in the chosen subset. In order to determine if the same relation holds for all missing cutting patterns as well, one can solve the following *knapsack problem (KSP)*

$$\begin{aligned} \max\ yb & \\ ab \leq L & \\ x_i \in \{0,1\}, & \quad \text{for all } i \in \{1,2,\ldots,m\} \end{aligned} \tag{16}$$

for which an effective pseudo-polynomial dynamic programming algorithm exists, as is pointed out in section 4.7. If the maximum is at most 1, the partial solution is optimal for the complete problem, otherwise the optimum pattern $b_1, b_2, \ldots, b_m$ found by the knapsack algorithm is appended as a new column to the formulation. This approach is called *column generation*. The process of evaluating the missing columns is called *pricing* in linear programming terminology, and embedding the method into an enumerative frame leads to a *branch-and-price-algorithm*. Cutting and pricing can be combined (and they are for example combined in all published state-of-the-art algorithms for the optimum solution of the traveling salesman problem) and this methodology is called *branch-and-cut-and-price*.

4.6 Primal Methods

A test set of an integer program of the form $\max\{cx \mid Ax \leq b,\ x \text{ integer}\}$ is a finite set T of vectors, such that for every non-optimal, feasible point x there exists a vector $t \in T$ such that $x + t$ is feasible and $c(x + t) > cx$. Such a vector t is called an *augmenting vector* for x. If a test set is available, a *primal integer programming approach* may consist of computing a sequence of feasible integer solutions with improving objective function value until an optimum is reached. In each iteration, either an augmenting element of the test set has to be produced or it must be decided that there no such vector exists (thus proving optimality). Notice the analogy to the cutting plane aproach, which is a "dual" method.
Test sets for integer programs are special integral bases for the integer points in certain convex sets. Such bases play an important role in several areas of mathematics such as algebra, number theory, combinatorics, and geometry. However, they are huge and hard to compute in general. Today's algorithms for the computation of a test set are unsuitable for large problems, yet this line of research has recently led to an algorithm that is sometimes capable of solving large-scale integer programs in a reasonable period of time. The key issue is that this new algorithm uses the power of LP-duality in combination with the theory of integral bases.
These recent developments make it likely that the primal approach will become a serious competitor in the solution of hard (mixed) integer programs in the future.

4.7 Dynamic Programming

Dynamic programming is a principle that is primarily used for the optimization of time discrete processes, but can also be employed for general discrete optimization problems. Consider a process that will run over T time periods. The process starts at time 0 in state s_0, proceeds to state s_1, then to s_2, etc. until it eventually reaches its final state s_T. At the start of each time period t a decision x_t has to be taken. The state of the process in period t only depends on x_t and on the previous state s_{t-1}. It is determined by the evaluation of a function $\phi(s_{t-1}, x_t)$. Each period contributes the amount g_t to the objective function which also only depends on x_t and s_{t-1}. Feasible decisions and states are given by the sets X_t and S_t. Then the optimization problem can be formulated as

$$\begin{aligned} z^* := \max \sum_{t=1}^{T} & g_t(s_{t-1}, x_t) \\ s_t = \phi(s_{t-1}, x_t), &\quad \text{for all } t \in \{1, 2, \dots, T\}, \\ x_t \in X_t, &\quad \text{for all } t \in \{1, 2, \dots, T\}, \\ s_t \in S_t, &\quad \text{for all } t \in \{1, 2, \dots, T\}. \end{aligned} \tag{17}$$

For $k = 1, 2, \dots, T$ one can define the optimal completion of the process when started in state s_{k-1} by

$$\begin{aligned} z_k(s_{k-1}) := \max_{x_k,\dots,x_T} \sum_{t=k}^{T} & g_t(s_{t-1}, x_t) \\ s_t = \phi(s_{t-1}, x_t), &\quad \text{for all } t \in \{k, k+1, \dots, T\}. \end{aligned} \tag{18}$$

In particular, we have $z^* = z_1(s_0)$ for the given initial state s_0. The crucial observation for the design of an algorithm is the validity of the following recursion

$$z_k(s_{k-1}) = \max_{x_k}\{g_k(s_{k-1}, x_k) + z_{k+1}(\phi(s_{k-1}, x_k))\} \tag{19}$$

for $k = T, T-1, \ldots, 1$ (where $z_{T+1}(\ ) = 0$). Based on this recursion one has a simple scheme for computing an optimum solution, provided that the set of possible states is not too large. Time as well as storage complexity is $O(|S| \cdot |T|)$, where $|S|$ is the number of possible states.
Consider for example a dynamic programming approach for the knapsack problem $\max\{cx \mid ax \leq b, x \in \{0,1\}\}$ with integer data. It can be considered as a process where in every time period $t = 1, 2, \ldots, n$ it has to be decided whether object t is selected for the knapsack or not, the state of the process being the remaining capacity of the knapsack. Using the notation above gives $s_0 = b$ and $\phi_t(s_{t-1}, x_t) = s_{t-1} - a_t x_t$. Furthermore $X_t = \{0,1\}$, $S_t = \{0, 1, \ldots, b\}$, and simply $g_t(s_{t-1}, x_t) = c_t x_t$ since the contribution to the objective function does not depend on the state. Notice that the running time $O(n \cdot b)$ is not polynomial in the input size. However, if b is not too big, then dynamic programming is a reasonable approach to solve the problem. In particular, if the value of b is bounded by a polynomial in n, then the dynamic programming algorithm is a polynomial one.
But also if the state space is large, dynamic programming can be tuned to work. By using exact *dominance rules* or heuristics one can try to eliminate inferior states during the recursive calculation (possibly at the expense of losing optimum solutions), see: Dynamic Programming.

4.8 Heuristics

Heuristics are not a topic of this article. Their importance has been claimed in the discussion of branch-and-bound algorithms. Whereas for pure combinatorial optimization problems there is a wide range of heuristic approaches (local search, genetic algorithms, tabu search, simulated annealing, etc.) at hand (see: Global Optimization and Meta-Heuristcs), this is not the case for general integer programming problems.
For finding good feasible solutions for general IPs, usually two principles are pursued. Firstly it is tried to round an infeasible solution. The difficulty is that there are many rounding possibilities and, moreover, none of them might lead to a feasible solution. Secondly it is tried to initially go deep into the branch-and-bound tree without putting too much effort in solving good relaxations in order to find a feasible solution early and, after that, giving priority to finding tight relaxations in order to finish the algorithm. Both approaches still are in the state of experimentation and neither of them can guarantee success. The area of developing heuristics for general integer problems is still an open one.

5 Software

There has been a lot of recent progress in software design for solving mixed integer programming problems and there is a number of very good software systems representing the scientific state-of-the-art, most of them commercial. Whereas previously algorithms were mainly branch-and-bound algorithms based on the canonical linear programming relaxation, current software uses sophisticated cutting plane generation routines and is much more powerful, with the consequence that mixed integer models can now be used in many more situations than before. In addition, software systems have aided the design of branch-and-cut-and-price algorithms for combinatorial optimization problems to various degrees since they became popular in the early eighties. Many of them are free.

References

[1] Cook W., Cunningham W., Pulleyblank W., Schrijver, A. (1998) Combinatorial Optimization. [This is a very good textbook on combinatorial optimization].

[2] Dell’Amico M., Maffiolo F., Martello S. (eds) (1997) *Annotated Bibliographies in Combinatorial Optimization*. John Wiley & Sons. [This gives a good guide to references in discrete optimization].

[3] Jünger M., Reinelt G., Thienel, S. (1995) *Practical Problem Solving with Cutting Plane Algorithms in Combinatorial Optimization* in: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 20: Combinatorial Optimization*, 111–152. [This describes the techniques for implementing branch-and-cut algorithms]

[4] Padberg M. (1995) *Linear Optimization and Extensions*, volume 12 of *Algorithms and Combinatorics*. Springer. [This gives a complete discussion of linear programming and polyhedral theory].

[5] Schrijver A. (1986) *Theory of Linear and Integer Programming*. John Wiley & Sons. [This discusses the theoretical foundations of integer programming in depth].

[6] Nemhauser G.L., Wolsey L.A. (1988) *Integer and Combinatorial Optimization*. John Wiley & Sons. [This treats theoretical as well as practical aspects of integer and combinatorial optimization].

[7] Weismantel R. (1998) *Test sets of integer programs*, MMOR 47, 1–37. [This is a survey about augmentation algorithms and test sets].