# Integer (Linear) Optimization

Egon Balas[1] and Matteo Fischetti[2]

An amazing variety of activities and situations can be adequately modeled as linear optimization problems, also known as linear programs (LPs), or convex nonlinear optimization problems. Adequately means that the degree of approximation to reality that such a representation involves is acceptable. However, as the world that we inhabit is neither linear nor convex, the most common obstacle to the acceptability of these models is their inability to represent nonconvexities or discontinuities of different sorts. This is where integer programming comes into play: it is a universal tool for modeling nonconvexities of various kinds.

To illustrate, suppose we have to put together an optimal production plan for a factory whose capacity limitations can be expressed through a system of linear constraints. This can be formulated as a linear program, but when the latter gets solved, the resulting plan is likely to be unacceptable because it violates a number of other conditions, like: threshold constraints (there is a minimum amount under which it is not worth producing an item), implications (if you are going to produce A, you need to also produce B), mutual exclusivity (you either do A or do B, but not both), precedence constraints (action A must precede action B by a certain amount of time), etc. It should be clear that conditions of this type are in no way exceptional. On the contrary, their presence is pervasive in most real world situations.

A linear (nonlinear) optimization problem whose variables are restricted to integer values is called a linear (nonlinear) integer (or discrete) optimization problem, or simply an *integer program* (IP, linear unless otherwise stated). If only some of the variables are restricted to integer values, we have a *mixed integer program* (MIP). Such a problem can be stated as $\min\{c^T x : Ax \geq b, x \geq 0, x_j \text{ integer for } j \in N_1 \subseteq N\}$, where $A$ is a given $m \times n$ matrix, $c$ and $b$ are given vectors of conformable dimensions, $N := \{1, \ldots, n\}$ and $x$ is a variable $n$-vector. The 'pure' integer program (IP) is the special case of MIP when $N_1 = N$.

Integer programming or integer optimization as a field started in the mid-1950s. A number of excellent textbooks are available for its study [39, 42, 45].

## Scope and Applicability

Applications of integer programming abound in all spheres of decision making. Some typical real-world problem areas where integer programming is particularly useful as a modeling

[1]Carnegie Mellon University, Tepper School of Business, 5000 Forbes Avenue, Pittsburgh PA 15213, `eb17@andrew.cmu.edu`

[2]DEI, University of Padova, via Gradenigo 6/A, 35136 Padova (Italy), `matteo.fischetti@unipd.it`

To appear in *Advances and Trends in Optimization with Engineering Applications*, T. Terlaky, M. F. Anjos, and S. Ahmed (editors), MOS-SIAM Book Series on Optimization, SIAM, Philadelphia, 2017 (print ISBN 9781611974676, ebook ISBN 9781611974683).

tool, include facility (plant, warehouse, hospital, fire station) location; scheduling (of personnel, production, other activities); routing (of trucks, tankers, aircraft); design of communication (road, pipeline, telephone, computer) networks; capital budgeting; project selection; analysis of capital development alternatives. Various problems in science (physics: the Ising spin glass problem; genetics: the sequencing of DNA segments) and medicine (optimizing tumor radiation patterns) have been successfully modeled as integer programs. In engineering (electrical, chemical, civil, mechanical, biological and medical) the sphere of applications is growing steadily, see below.

By far the most important special case of integer programming is the (pure or mixed) *0-1 programming problem*, in which the integer-constrained variables are restricted to 0 or 1. This is so because a host of frequently occurring nonconvexities, such as the ones listed above, can be formulated via 0-1 variables. For instance, the threshold condition mentioned above (the amount $x$ produced cannot be less than some quantity $q$), i.e. $x \leq 0$ or $x \geq q$, can be expressed through the inequalities $q\delta \leq x \leq Q\delta$, $\delta \in \{0,1\}$, with the 0-1 variable $\delta$ enforcing the threshold: if $\delta = 0$, $x = 0$; if $\delta = 1$, $q \leq x \leq Q$, i.e. $x$ will be between the threshold $q$ and a large enough upper bound $Q$.

Next we present a few well-known pure and mixed integer models.

The *fixed charge problem* asks for the minimization, subject to linear constraints, of a function of the form $\sum_i \gamma_i(x_i)$, with

$$\gamma_i(x_i) := \begin{cases} f_i + c_i x_i & \text{if } x_i > 0, \\ 0 & \text{if } x_i = 0. \end{cases}$$

As an example, consider the problem of installing a communications network whose arcs can have different capacities at different costs, with the objective of minimizing the total cost while satisfying certain overall capacity requirements. Whenever $x_i$ is bounded by $U_i$ and $f_i > 0$ for all $i$, such a problem can be restated as a (linear) MIP by setting for all $i$

$$\begin{aligned} &\gamma_i(x_i) = c_i x_i + f_i y_i, \\ &x_i \leq U_i y_i, \\ &y_i \in \{0,1\}. \end{aligned}$$

Clearly, when $x_i > 0$ then $y_i$ is forced to 1, and when $x_i = 0$ the minimization of the objective function drives $y_i$ to 0.

The *facility location problem* consists of choosing among $m$ potential sites (and associated capacities) of facilities to serve $n$ clients at a minimum total cost:

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{m} f_i y_i \\ & \sum_{i=1}^{m} x_{ij} = d_j, \quad j = 1, \ldots, n, \\ & \sum_{j=1}^{n} x_{ij} \leq a_i y_i, \quad i = 1, \ldots, m, \\ & x_{ij} \geq 0, \quad i = 1, \ldots, m; j = 1, \ldots, n, \\ & y_i \in \{0,1\}, \quad i = 1, \ldots, m. \end{aligned}$$

Here $d_j$ is the demand of client $j$, $a_i$ is the capacity of a potential facility to be located at site $i$, $c_{ij}$ is the per-unit cost of shipments from facility $i$ to client $j$, and $f_i$ is the fixed cost of opening a facility of capacity $a_i$ at location $i$. In any feasible solution, the indices $i$ such that $y_i = 1$ designate the chosen locations for the facilities to be opened, while the continuous variables $x_{ij}$ represent the quantity shipped from facility $i$ to client $j$.

Variants of this problem include the *warehouse location problem* (which considers cheap bulk shipments from plants to warehouses and expensive packaged shipments to retailers) and various *emergency facility location* problems (where one chooses locations to minimize the maximum distance traveled by any user of a facility, rather than the sum of travel costs).

The *knapsack problem* is an integer program with a single constraint:

$$\max\{c^T x : a^T x \leq b, x \geq 0 \text{ integer}\},$$

where $c$ and $a$ are positive $n$-vectors, while $b$ is a positive scalar. It represents the choices one faces when trying to fill a container of capacity $b$ with objects $j$ of volume $a_j$ and value $c_j$ in such a way as to maximize the value of the objects included.

When the variables are restricted to 0 or 1, we have the *0-1 knapsack problem.*

A variety of situations can be fruitfully modeled as *set covering problems*: Given a set $M$ and a family of weighted subsets $S_1, \ldots, S_n$ of $M$, find a minimum-weight collection $C$ of subsets whose union is $M$. If $A$ is a 0-1 matrix whose rows correspond to the elements of $M$ and whose columns are the incidence vectors of the subsets $S_1, \ldots, S_n$, and $c$ is the $n$-vector of subset-weights, the problem can be stated as

$$\begin{array}{ll} \min & c^T x \\ & Ax \geq 1 \\ & x \in \{0,1\}^n, \end{array}$$

where the right-hand side of the inequality is the $m$-vector of 1s. This model and its close relative, the *set partitioning problem* (in which $\geq$ is replaced by $=$) has been (and is being) widely used in airline, bus, and train crew scheduling (each row represents a leg of a trip that has to be covered; each column stands for a potential duty period of a crew). Another application is in medical diagnostics: each column represents a diagnostic test, each row stands for a pair of diseases, with a 1 in column $j$ if the pair's reactions to the tests are different, and a 0 if they are the same; the goal being to select a minimum-cost battery of tests guaranteed not to yield identical outcomes for any two diseases.

Unlike in the case of linear optimization, integer optimization problems can typically be formulated in several different ways, and choosing a good formulation is of paramount importance. What should be the guiding choice criterion when comparing different formulations? In linear optimization, the criterion is typically the number of variables and constraints; the fewer of each, the better. In integer optimization the situation is very different. Because of the central role of branch-and-bound methods in solving integer programs (see below), whose efficiency depends on the strength of the bounds provided by the linear relaxation of the instance being solved, the leading criterion in choosing between different formulations is the tightness of the linear relaxation provided by each formulation. Thus in choosing between two formulations of a minimization problem, one of which has twice as many variables and/or constraints than the other, but has an optimum over its linear relaxation higher than the other, the formulation with the higher LP optimum is usually preferable.

## Combinatorial Optimization

A host of interesting combinatorial problems can be formulated as 0-1 programming problems defined on graphs, undirected or directed, vertex-weighted or edge-weighted. The joint study of these problems by mathematical programmers and computer scientists, starting from around 1960, has led to the development of the burgeoning field called *combinatorial optimization* [20]. Some typical problems of this field are: edge matching (finding a maximum-weight collection of pairwise non-adjacent edges) and edge covering (finding a minimum-weight collection of edges that together cover every vertex); vertex packing (finding a maximum-weight independent set, i.e. collection of pairwise non-adjacent vertices) and vertex covering (finding a minimum-weight collection of vertices that together cover every edge); maximum clique (finding a maximum cardinality complete subgraph) and minimum vertex coloring (partitioning the vertices into a minimum number of independent sets, i.e. coloring the vertices with a minimum number of colors such that all adjacent pairs differ in color); the traveling salesman problem (finding a cycle of minimum total edge-weight that meets every vertex).

We will briefly discuss two of the above problems, which in a sense span the universe of combinatorial optimization. At one end of the spectrum, the *matching problem* on a graph $G = (V, E)$ can be stated as

$$\max\{x(E) : x(\delta(v)) \leq 1, v \in V, x \geq 0 \text{ integer}\},$$

where $x(F) = \sum\limits_{e \in F} x_e$ for $F \subset E$ and $\delta(v)$ is the set of edges incident with $v$. Here $x_e = 1$ if $e$ is in the matching, $x_e = 0$ otherwise. The weighted version of the problem asks for maximizing $\sum_{e \in E} w_e x_e$, where $w_e$ is the weight of edge $e$. This problem has the nice property that the integrality condition can be omitted if the above nonnegativity and degree constraints are supplemented with the inequalities

$$x(\gamma(S)) \leq \frac{|S| - 1}{2} \text{ for all } S \subseteq V, |S| \text{ odd},$$

where $\gamma(S)$ is the set of edges with both ends in $S$. In other words, the above 'odd set inequalities', along with the nonnegativity and degree constraints, fully describe the convex hull of incidence vectors of matchings. The discovery of this remarkable phenomenon in the mid-1960s (due to J. Edmonds [22]) has started a massive pursuit of facets of the convex hull of other combinatorial polyhedra, and can be viewed as the inaugural step in the development of the field called *polyhedral combinatorics*. It can also be viewed as a precursor of the theory of $\mathcal{NP}$-completeness in the early-to-mid 1970s ([19, 32]), which brought fundamental clarity to the issue of computational complexity, in that it classified problems into polynomially solvable ones (i.e., solvable in time polynomial in the problem size, like linear programming and the matching problem mentioned above), and those called $\mathcal{NP}$-complete, for which no polynomially bounded procedure is known (and is never likely to be found, since if one of them were polynomially solvable, then all of them would be).

At the other end of the spectrum, one of the hardest and most thoroughly investigated combinatorial optimization problems is the *traveling salesman problem* (TSP) already mentioned, in which a salesman is looking for a cheapest tour of $n$ cities, given the cost of travel between all pairs of cities. This problem, $\mathcal{NP}$-complete according to the above classification, is the prototype model for situations dealing with the optimal sequencing of objects (e.g., items to be processed on a machine in the presence of sequence-dependent setup costs). The standard formulation on a complete directed graph with node set $N$ and arc costs $c_{ij}$ is

$$\begin{array}{ll}
\min & \sum_{i\in N}\sum_{j\in N\setminus\{i\}} c_{ij}x_{ij} \\
\text{s.t.} & \sum_{j\in N\setminus\{i\}} x_{ij} = 1, \quad i \in N, \\
& \sum_{i\in N\setminus\{j\}} x_{ij} = 1, \quad j \in N, \\
& \sum_{i\in S}\sum_{j\in S\setminus\{i\}} x_{ij} \le |S| - 1, \; S \subset N, 2 \le |S| \le n-1, \\
& x_{ij} \in \{0,1\}, \quad i,j \in N, \quad i \ne j.
\end{array}$$

The first two sets of equations define an assignment problem whose solutions are spanning unions of directed cycles. The third set, consisting of inequalities called subtour elimination constraints, exclude all cycles with fewer than $n = |N|$ arcs. The number of solutions – tours – is factorial in the number of nodes. The TSP has become a test bed for the development of, and experimentation with, various approaches to combinatorial optimization [3].

## Solution Methods

Unlike linear programs, integer programming problems, including 0-1 programming and most combinatorial optimization problems, are $\mathcal{NP}$-complete. The difficulty in solving integer programs lies in the nonconvexity of the feasible set, which makes it impossible to establish global optimality from local conditions. The two principal approaches to solving integer programs try to circumvent this difficulty in two different ways.

The first approach, which until the late 1980s was the standard way of solving integer programs, is *enumerative* (branch and bound, implicit enumeration). It partitions the feasible set into successively smaller subsets tied together as nodes of a branch and bound tree, calculates bounds on the objective function value over each subset, and uses these bounds to discard certain subsets (nodes of the tree) from further consideration. The lower bounds (in a minimization problem) typically come from solving the linear relaxation corresponding to the given node (i.e., the linear program obtained by removing the integrality condition), the upper bounds come from integer solutions found at some of the nodes. The procedure ends when each subset has either produced a feasible solution, or was shown to contain no better solution than the one in hand. The efficiency of the procedure depends crucially on the strength of the bounds. An early prototype of this approach is due to A.H. Land and A.G. Doig [35], another one is [4].

The second approach, known as the *cutting plane method*, is a convexification procedure: it approximates the convex hull of the set of feasible integer points by a sequence of inequalities that cut off (hence the term 'cutting planes') part of the linear programming feasible set, without removing any feasible integer point. The first finitely convergent procedure of this type, which uses modular arithmetic applied to the rows of the optimal simplex tableau to derive valid cutting planes for pure integer programs, is due to R.E. Gomory [29] who later extended it to the mixed integer case. V. Chvátal [18] has shown that the procedure can be viewed as one of *integer rounding*, in which positive multiples of $Ax \geq b$, $x \geq 0$ are added up and the coefficients of the resulting inequality are rounded down to the nearest integer. The resulting inequalities form the elementary closure of $Ax \geq b$, $x \geq 0$. The procedure can then be applied to the elementary closure, and so on. The number of times the procedure needs to be iterated in order to obtain the convex hull of integer points within a polyhedron is called the Chvátal rank of the given polyhedron. No bound is known on the Chvátal rank of an arbitrary polyhedron. By contrast, the matching polyhedron has Chvátal rank one, since the so called odd set inequalities can be obtained from the degree inequalities by integer rounding. The Gomory-Chvátal procedure has been extended to mixed integer programming and has been enhanced by the use of subadditive functions and group theory [29, 30].

A different approach, originating in the early 70s, uses geometric concepts and convex analysis. Given a basic fractional solution $\bar{x}$ to the LP relaxation of a MIP, the inequalities that are tight at $\bar{x}$ form a cone $C(\bar{x})$ (see Figure 1). If $S$ is any convex set whose interior contains $\bar{x}$ but no feasible integer points, then the hyperplane through the intersection points of the $n$ extreme rays of $C$ with the boundary of $S$ defines an inequality that cuts off $\bar{x}$ but no feasible integer point. Such an inequality is an intersection cut [5]. If the integer-constrained variables are basic in the optimal solution with one of them, say $x_k$, fractional, and $S := \{x : \lfloor \bar{x}_k \rfloor \leq x_k \leq \lceil \bar{x}_k \rceil\}$ (where $\lfloor\ \rfloor$ and $\lceil\ \rceil$ means rounding down and rounding up), then the intersection cut from the convex set $S$ is the Gomory cut from the row of the simplex tableau associated with $x_k$.

An intersection cut from a convex polyhedron like $S$ can also be viewed as a cut from the disjunction that results from reversing the inequalities defining $S$, i.e., from the condition $x_k \leq \lfloor \bar{x}_k \rfloor$ or $x_k \geq \lceil \bar{x}_k \rceil$ which has to be satisfied by any feasible integer point. Viewed this way, the condition can be strengthened by amending each term of the disjunction with all inequalities valid for both, which gives rise to a condition of the form

$$\begin{pmatrix} Ax & \geq & b \\ x_k & \leq & \lfloor \bar{x}_k \rfloor \end{pmatrix} \quad \vee \quad \begin{pmatrix} Ax & \geq & b \\ x_k & \geq & \lceil \bar{x}_k \rceil \end{pmatrix}.$$

Such a disjunction is a union of two polyhedra, and this has led to the study of disjunctive programming [6, 7], or linear programming with logical conditions (conjunctions, disjunctions and implications involving inequalities). In this approach, which uses the tools of convex analysis, like polarity and projection, 0-1 programming (pure or mixed) is viewed as optimization over the (nonconvex) union of (convex) polyhedra, i.e. a set of the form

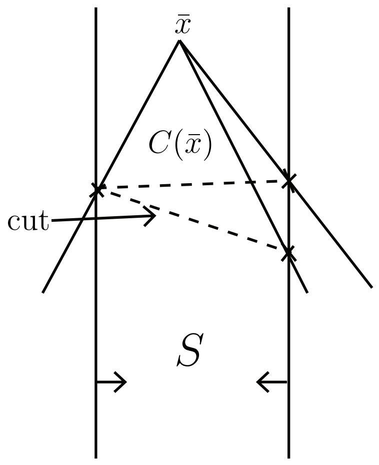

Figure 1:

$\cup_{i\in Q} P_i$, where $P_i = \{x : A^i x \geq b^i\}$, $i \in Q$. There is a compact formulation of the convex hull $P_D :=$ conv $\cup_{i\in Q} P_i$ in a higher-dimensional space, whose projection onto the original space yields all the valid cutting planes. Thus, from a classical theorem of the alternatives known as the Farkas Lemma, it follows that $\alpha^T x \geq \beta$ is a valid inequality for $\cup_{i\in Q} P_i$ if and only if $\alpha^T \geq u_i^T A^i$ and $\beta \leq u_i^T b^i$ for some $u_i \geq 0$, $i \in Q$. A central result of this approach is that an important class of disjunctive programs, called *facial*, which includes pure and mixed 0-1 programs, are *sequentially convexifiable*. For a 0-1 program (pure or mixed) with $n$ 0-1 variables and a linear programming relaxation $P_0$, this means that one can impose the 0-1 condition on $x_1$ and generate the convex hull $P_1$ of $P_{00} \cup P_{01}$, where $P_{00} := \{x \in P_0 : x_1 = 0\}$, $P_{01} := \{x \in P_0 : x_1 = 1\}$. Then impose the 0-1 condition on $x_2$ and generate the convex hull $P_2$ of $P_{10} \cup P_{11}$, where $P_{10} := \{x \in P_1 : x_2 = 0\}$, $P_{11} := \{x \in P_1 : x_2 = 1\}$; etc. At the end of $n$ steps, the convex hull $P_n$ of $P_{n-1,0} \cup P_{n-1,1}$ turns out to be the convex hull of $\{x \in P_0 : x_j \in \{0, 1\}, j = 1, \ldots, n\}$. This property does not hold for arbitrary integer programs, and is thus a main distinguishing feature of 0-1 programs. If one defines the disjunctive rank of a polyhedron as the number of times the above procedure has to be iterated in order to generate all of its facets, it follows that an arbitrary 0-1 programming polyhedron has disjunctive rank $n$.

## Revolution in the state of the art

Integer programming was recognized early on as a miraculous tool which could be used to model almost any problem; however, the result of such an exercise was usually a toy model which provided insights but could not be actually solved. Of the two approaches described above, branch and bound was relatively straightforward to implement, and for about three decades it was the only one practically used to solve some real world problems. However, the size of these problems – apart from some special structures – was limited to 30-50 variables. As to the cutting plane approach, the attempts at its computational implementation were unsuccessful. As one would generate a cut, add it to the constraints and reoptimize the resulting linear program, then repeat this cycle, numerical problems would arise and prevent the solution of even small problems. So for three decades cutting planes were the object of intense theoretical research, but the only practical tool for solving integer programs were the branch and bound codes, commercial versions of which appeared by the seventies. This situation persisted until the late eighties; then, roughly in the 15-year period between 1990-2005, a true revolution occurred in the state of the art of integer optimization. A simple measure of this is the following. Over the years, a collection of integer programming instances was developed under the name of MIPLIB [33]. Whereas earlier most MIPLIB instances remained unsolved, after the turn of the century most of them including many new ones, got solved. Whereas earlier general IP's with more than 50 variables could not be solved, nowadays instances with thousands of variables are routinely solved (see below for some details).

How did this revolution occur? Several factors played a role. Among them, faster computers, more efficient LP codes, various heuristics, more sophisticated branching rules; but the crucial factor was the embedding of cutting planes into a branch-and-bound context. Next we briefly outline how this came about.

The key impetus came from a revival of the disjunctive programming approach outlined above and its implementation into an efficient computational tool called *lift-and-project* [9]. The name conveys the idea of a higher dimensional representation of the convex hull (lifting), which is then projected back to generate cutting planes. In the meantime, L. Lovász and A. Schrijver [37] (see also [44]) developed a closely related procedure which derives higher-dimensional representations of a 0-1 programming polyhedron by multiplying the constraint set of $P_0$ with the inequalities $x_j \geq 0$ and $1 - x_j \geq 0$, $j \in N$, then linearizing the resulting quadratic forms, and projecting them back into the original space. As in the disjunctive programming approach, $n$ iterations of this procedure yield the convex hull of the 0-1 programming polyhedron. However, the quadratic forms obtained during the procedure can also be used to derive *positive semidefiniteness constraints* that are stronger than the inequalities obtained by linearization.

Semidefiniteness constraints aside, a streamlined version of the Lovász-Schrijver procedure, in which $P_0$ is multiplied at every iteration by just one pair of inequalities $x_j \geq 0$, $1 - x_j \geq 0$, rather than by all pairs, was shown in [9] to be equivalent to the disjunctive programming procedure for 0-1 polyhedra, in that the linearized version of the quadratic constraints obtained by multiplication is exactly the same as the higher-dimensional representation of the convex hull used in disjunctive programming [6, 7]. The paper [9] also showed how to use a cut generating linear program (CGLP) to obtain lift-and-project (or disjunctive) cuts that are deepest in a well defined sense. Most importantly, these cuts can be generated in a subspace, i.e. using only a subset of the variables, and then lifted to the full space. This has opened the door to combining the enumerative and convexifying approaches into a *branch and cut procedure*, which generates cutting planes as long as they 'work', but branches whenever the cut generating 'stalls'. This was done earlier for the special case of the TSP [40], but its extension to general MIP's was made possible by showing that cuts generated at a node of the search tree can be lifted to be valid at any node. The outcome was a robust procedure, considerably more efficient than either a branch and bound or a cutting plane algorithm by itself [10].

Experimentation with this procedure showed that the loss of accuracy and related numerical problems that used to accompany the recursive cut generating procedure of reoptimizing the LP after the addition of every cut, can be substantially mitigated by generating *rounds* of cuts, one from each row whose integer-constrained basic variable is fractional, before reoptimizing. Finally, as a consequence of these experiments, it turned out that Gomory cuts, when embedded in a branch and bound procedure and generated in rounds, could also be computationally tractable [11]. Since these are easier to generate, they were the first ones to make it into the commercial codes in the 1990's. The stronger but more expensive lift-and-project cuts made it into the commercial codes after a way was found [12] to solve the CGLP indirectly, without actually setting it up, by pivoting in the LP simplex tableau. Other cuts used include mixed integer rounding, knapsack cover, flow cover, clique cuts.

To give the reader an idea of the extent of the revolution in the computational capability of integer programming solvers it is worth citing a few numbers. According to the developers of the commercial code CPLEX [15], a comparison of the performance of CPLEX 1.2 (1991) and CPLEX 11.0 (2007) on 1,852 real-world instances of linear mixed integer programs yielded the following results. The speedup in computing time during those 16 years was 29,530-fold. In other words, the 2007 version of CPLEX ran on the average almost 30,000 times faster than the 1991 version, which corresponds roughly to a doubling of the speed every year. As a result, whereas in 1991 only 15% of the instances attempted could be solved, in 2007 69% of the instances were solved. After 2007, the improvements in computing power continued at a somewhat slower, but still steady rate: the 2013 version (5.6) of the commercial code GUROBI is 21 times faster than the 2009 version (1.0) [14].

## Heuristics

While solving integer optimization problems exactly was a hard nut to crack and it took more than thirty years to reach a degree of success that made large scale commercial applications possible, various heuristic procedures were developed over the years with the goal of finding more or less acceptable approximate solutions. By more or less acceptable we mean practical, differing from the optimum usually by a few percentage points, as opposed to the theoretical acceptability offered by a guaranteed bound which is a polynomial function of the optimum. Not surprisingly, the type of heuristics that enjoyed the earliest successes were those devoted to solving special structures. For instance, the notoriously hard traveling salesman problem, a model for sequence-optimization problems of various types, was quite successfully attacked by a type of local improvement heuristic based on interchanging some in-tour arcs with out-of-tour arcs (2-,3-,$k$-interchange), which culminated in the *variable-depth interchange* heuristic of Lin and Kernighan [36], whose latest version [31] is able to find good approximate solutions to instances with hundreds of thousands of variables in a matter of seconds. Various combinations of primal and dual greedy heuristics, Lagrangean relaxation and subgradient optimization were able to solve large set covering and partitioning problems arising in airline and railway crew scheduling [8, 17]. A job shop scheduling heuristic based on a disjunctive graph formulation and the idea of repeatedly identifying the bottleneck machine and tackling it locally, called the *shifting bottleneck procedure* [2], has become a basic staple of operations scheduling in various job-shop environments (see chapter 5 of [41]).

By the 1990s a number of metaheuristics have been proposed, like for instance *tabu search* [28], i.e. procedures based on some general heuristic principle, whose various incarnations are specific heuristics for certain problem classes. One such general principle is that of *neighborhood search*. Elements of such heuristics made their way into the leading mixed integer solvers and were applied locally at various stages of the search procedure to speed up its convergence.

One of the procedures that proved to be successful in practice was the *feasibility pump* (FP), originally proposed for the 0-1 case [24] and then extended to general MIP's [13]. It is based on the observation that a feasible MIP solution is a point $x$ of $P$ that coincides with its rounding. Replacing "coincides" with "is as close as possible" leads to the following iterative scheme, to be described for the sake of simplicity, for pure 0-1 IP's. FP works with a pair of point $(x^*, x')$, with $x^*$ in $P$ and $x'$ integer, that are iteratively updated with the aim of bringing them as close to each other as possible, i.e. minimizing

$$\Delta(x^*, x') := \sum_{j \in N} |x^*_j - x'_j| = \sum_{j \in N : x'_j = 0} x^*_j - \sum_{j \in N : x'_j = 1} (1 - x^*_j).$$

To be more specific, one starts with any $x^*$ in $P$, e.g. an optimal LP solution, and initializes an integer $x'$ as the rounding of $x^*$. At each FP iteration, called a pumping cycle, $x'$ is fixed and one finds a point $x^*$ in $P$ which is as close to $x'$ as possible, by solving an auxiliary LP. If $\Delta(x^*, x') = 0$, $x^*$ is integer and we are done; otherwise $x'$ is replaced by the rounded $x^*$ in an attempt to bring them closer, and the process is iterated. Several variants and extensions have been proposed and implemented [1, 16, 27].

A new generation of MIP heuristics emerged in the late 1990s. Their hallmark is the use of a "black-box" external MIP solver to explore a solution neighborhood defined by invalid linear constraints. The use of an exact MIP solver inside a MIP heuristic may appear naive at first glance, but it turns out to be effective in the cases where the added invalid constraints lead to a structural simplification of the MIP at hand and allow, e.g., for a more powerful instance preprocessing and/or for extensive node pruning. The *local branching* (LB) scheme of [25] appears to be the first method embedding a MIP solver within a general MIP heuristic framework. Suppose a feasible *reference solution* $\tilde{x}$ of a 0-1 MIP is given, and one aims at finding an improved solution that is "not too far" from $\tilde{x}$. To this end, one can define the *k-OPT neighborhood* of $\tilde{x}$ as the set of the MIP solutions satisfying the invalid *local branching constraint* $\Delta(x, \tilde{x}) \leq k$ for a small parameter $k$ (typically, $k = 10$ or $k = 20$), and explore it by means of an external MIP solver, often heuristically, i.e., within a prefixed number of branch-and-bound nodes. The method is in the spirit of local search metaheuristics and in particular of *Large Neighborhood Search* (see, e.g., [43]), with the novelty that neighborhoods are obtained through invalid cuts to be added to the original MIP model. Diversification cuts can be defined in a similar way, thus leading to a flexible toolkit for the definition of metaheuristics for general MIP's.

The *Relaxation Induced Neighborhood Search* (RINS) framework of [21] also uses the (heuristic) solution of a simplified MIP through an external MIP solver as a main ingredient, but extends the idea by taking the solution of the LP relaxations into account. At specified nodes of the branch-and-bound tree, the current LP relaxation solution $x^*$ and the incumbent $\tilde{x}$ are compared and all integer-constrained variables that agree in value are fixed and removed, and the solution of the resulting sub-MIP is attempted by invoking the MIP solver itself with a tight time/node limit.

## The Impact of Contemporary MIP Technology

The revolution in the state of the art of MIP outlined above has brought this powerful modeling tool to bear on an increasing array of practical problems in industry, transportation, energy, finance, healthcare and a host of other activities. The annual competition run by INFORMS for the prestigious Franz Edelman Award attracts companies from across the globe. The purpose of the Award is to recognize and reward outstanding examples of real world applications of Operations Research and Management Science techniques. Since the launching of the annual competitions, cumulative benefits from the finalists' projects (of which there are six per year) have exceeded 210 billion dollars. The most frequently used technique in these applications turns out to be mixed integer programming. According to G. Nemhauser [38], 53% of all the finalists have used MIP techniques of one kind in their projects. Among the winners, the percentage of MIP users is even higher. All but two of the Edelman Prize awardees of the last decade use one form or another of MIP. To illustrate the use of these techniques, we briefly discuss two of the award winning projects.

The 2008 winner was Netherlands Railways for the entry "The New Dutch Timetable: The O.R. Revolution" [34]. By 2006, the volume of traffic on the Dutch passenger railway network had increased significantly; more and larger trains had been scheduled without changing the structure of the timetable, thus overloading the system and causing consumer nightmares. Operations researchers working with Netherlands Railways constructed an improved timetable. As a result, the percentage of trains arriving within three minutes of the scheduled time increased, commuter satisfaction improved, and the number of passengers grew. In 2007, this resulted in an additional annual profit of $60 million.

To construct a new timetable and its related resource schedules, a sequence of planning problems was defined and solved. As input, a railway line system was given. Then, the timetable was defined, including the detailed routings through the stations. Finally, rolling-stock and crew schedules were constructed. In each phase, MIP models were formulated and solved (often heuristically).

The model used for timetable generation describes the cyclic timetabling problem in terms of the periodic event scheduling problem (PESP) constraints. This is a generic model for scheduling a set of periodic events, such as the event times in a cyclic timetable. All PESP constraints are expressed as differences of event times. For example, the running time of a train from one station to another is the difference between the arrival time and the departure time. Similarly, the headway time is the difference between the departure times of two consecutive trains on the same track. All time differences are computed modulo 60 to reflect the timetables cycle of one hour. The timetabling problem was solved through CADANS (a proprietary constraint programming software for finding a first feasible PESP solution) and STATIONS (a MIP-based solver to find detailed routings through the stations) systems.

The goal in scheduling rolling stock is to allocate an appropriate amount of the appropriate rolling-stock type to each train in the given timetable. A constraint on the number of train units available is imposed—during peak hours, most trains will simultaneously require more units. A further complexity is that demand varies substantially during the day and on a line. The goal is to find a balance between three conflicting objectives in rolling-stock scheduling: (1) service, (2) efficiency, and (3) robustness. In this context, service means offering as many passengers as possible a seat. Efficiency aims at minimizing the amount of rolling stock and the number of rolling stock kilometers. Robustness is addressed by reducing the number of shunting movements and by having a line-based rolling-stock circulation. The resulting rolling stock problem was solved through a proprietary ROSA system built on top of a commercial MIP solver (Cplex).

Each train in the timetable requires a train driver and a number of conductors depending on the rolling-stock composition of the trains. Approximately 6,000 crew members (i.e., train drivers and conductors) operate from 29 crew bases throughout the Netherlands. Each crew member belongs to a specific crew base. A duty starts and ends in a crew base and describes the consecutive trips for a single crew member. For each day, a number of anonymous duties are first generated (crew scheduling phase), and then assigned to individual crew members on consecutive days (rostering phase).

The crew scheduling problem was solved by using TURNI, a commercial software based on the set covering (MIP) model. In such a model, there is a binary decision variable for each potential duty (1 if the potential duty is selected and 0 otherwise). The problem is then to select a subset of duties from a predetermined set of feasible ones such that it covers each trip by at least one duty, it satisfies all additional constraints at the crew-base level, and the total costs of the selected duties are as low as possible. As the number of feasible potential duties is extremely large, a column generation procedure is used, where a pricing model generates the feasible potential duties on the fly whenever they are needed. A typical workday includes approximately 15,000 trips for drivers and 18,000 for conductors. The resulting number of duties is approximately 1,000 for drivers and 1,300 for conductors. This leads to very difficult crew scheduling instances. Nevertheless, because of its highly sophisticated MIP algorithms, TURNI solves these cases in a few hours of computing time on a personal computer. Therefore, one can construct all crew schedules for all days of the week within just a few days.

In 2012, the Edelman prize was awarded to TNT Express for the entry “Supply Chain-Wide Optimization at TNT Express” [23]. TNT’s Global Optimization (GO) program initiative led to the development of optimization solutions to assist the operating units of TNT Express and improve their package delivery in road and air networks, based mostly on MIP-driven software. Over a seven year period, TNT Express used these optimization methods to save $207 million. In the framework of the GO program, three separate subprograms were developed and a portfolio of models, methodologies and mostly MIP-based tools was designed.

Subprogram 1, named TNT Express Routing and Network Scheduling (TRANS), was concerned with the optimization of routes for the transportation of packages and vehicle tours. A transportation route defines the sequence of hubs, from the depot of origin through to the destination depot, including scheduled times of arrival and departure at the hubs, that a package will visit. A tour describes the sequence of locations visited by a vehicle (and driver), including the times at which each location is visited. Because of the size of the networks that TNT Express operates, the problem was split into several subproblems, each supported by a specific module in TRANS: (i) a service capability analyzer determines the fastest feasible routes based on the prespecified movements in the network, and is based on a multicommodity flow problem in a time-space network solved through fast heuristics; (ii) a routing module generates a set of routes (not only the fastest) and assigns the packages to the movements of these routes by using a branch-and-bound algorithm to generate routes that meet the service requirements; (iii) an optimal paths module determines the optimal paths for each package, given the current infrastructure of depots and hubs, and is based on a network design MIP model.

Subprogram 2, named Tactical Planning in Pickup and Delivery (SHORTREC), was planned at the depot level and affected the first and last mile in the supply chain. Pickup and delivery accounts for more than 30% of operational costs, hence it was an important focus area of the GO program. At TNT Express, a round corresponds to a single vehicle starting at the depot, visiting customers in a certain sequence for collection or delivery of packages, and returning to the depot. Effectively organizing the whole process is challenging because millions of packages must be picked up and delivered each week. The optimization problem was to minimize the total pickup and delivery costs while meeting all service-level requirements. Constraints to be considered included vehicle capacity, service levels, driver regulations, and some softer restrictions to ensure repetitiveness in the rounds and workload balancing. An ad-hoc optimization software was designed, based on vehicle routing heuristics.

Subprogram 3 addressed the whole Supply Chain Optimization (DELTA Supply Chain). Because the air network forms a crucial part of TNT's global service offering, a supply chain optimization project was started to reduce aircraft use and to preserve future growth capabilities without worsening service. The DELTA Supply Chain model enabled TNT to optimize the complete supply chain for a fixed depot and hub infrastructure under varying volumes and ways of working (e.g., cutoff times, road and air transport). The model was aimed to using road transport rather than air transport because the former generally results in lower costs and CO2 emissions. For packages that can be shipped by road, the number of required movements was calculated based on the routings of the packages. For the packages that were unable to meet the service requirements via the road network, an air network was constructed by using a separate model to create a minimum-cost air schedule between the airports in the network. The model starts by assigning depots to the airports in the air network according to some heuristic criteria. Based on these assignments, the model determines the packages to be transported from the airport to the air hub and vice versa. Next, a MIP is solved to determine the minimum-cost air schedule. The model ensures that sufficient aircraft capacity is available to carry all the packages, and it balances the number of incoming and outgoing aircraft per aircraft type at each location. For airports, the MIP model includes the earliest permitted arrival or departure times, airport closing times, and the consideration that some airports do not permit multiple stops by TNT Express airplanes. With the road and air network complete, a binary integer programming model estimates the impact of the network movement arrival and departure times on the pickup-and-delivery cost. In a final step, the model calculates the total cost of the complete supply chain to support management decision making.

### How to use a MIP code

Several powerful MIP codes are nowadays available, including the commercial solvers IBM ILOG Cplex, Gurobi and FICO Xpress, as well as the open-source solvers SCIP, COIN-OR CBC, and GLPK (just to mention some of them). Each of these solvers can be accessed through the internet by using any search engine. Most solvers exploit multi-threading to take advantage of all the available processing-units (cores) of the CPU in use. In addition, a distributed version of the main MIP solvers is available, that distributes the computation over a set of independent computers in a cluster/network.

MIP codes can be used in command mode, meaning that a user can write his/her (integer) linear model into a text file according to suitable format (MPS or LP), and then read it in and solve by appropriate command. This approach is the easiest to apply as the solver is used as a black-box, with a limited control on its main parameters. In many practical cases, this is just what is needed to get a successful application.

A non-exhaustive list of the main MIP commands and parameters follows.

First of all, one has the *read* command to load the MIP model to be solved from a text file, and the *write* command to write the optimal solution found onto the screen or a text file. The *run/optimize* command invokes the MIP solver on the loaded instance, and the control is returned to the command mode when some termination conditions are reached.

The obvious termination condition is of course that the solver found a provable optimal solution. However, early termination can be enforced by changing some internal execution limits by using an appropriate *change parameter* command (to be executed before starting the optimization). The most common limits that a user can change include the overall computing time (say, in seconds), the number of branching nodes, the number of feasible solutions found (i.e., of incumbent updates), and the absolute/relative optimality gap (meaning that the execution terminates when the current-best integer solution is guaranteed to be sufficiently close to the true optimal one).

Modern MIP codes typically do not require an external fine-tuning of the parameters that determine the ‘aggressiveness’ of the internal heuristics and of the cutting-plane generation procedures, in the sense that the default mode modifies these parameters automatically, in an adaptive way. It should be noted however that the ultimate goal of the computation is assumed to be the proof of optimality of the best solution found. Hence the default mode can be inadequate in the hard cases where the user would prefer to use the MIP solver to find good feasible solutions within short computing times. In this ‘heuristic mode’ what matters is in fact the capability of quickly finding and improving feasible solutions, rather than proving the they are (almost) optimal. To this end, one can consider increasing the aggressiveness of the internal heuristics, by increasing the parameter controlling the *heuristic frequency* so as to invoke them every, say, 1 or 10 branching nodes. Moreover, some MIP solvers implement a final 'clean-up' post-processing of the best solution, which is possibly improved by using ad-hoc refining procedures. This final refining step is not applied by default, but it may be worth trying it with a short time limit. Deactivation of all cuts (through a suitable parameter that controls the aggressiveness of each cutting-plane generation function) can sometimes be useful in the heuristic mode—note however that cuts can be beneficial for heuristics too.

Recent MIP codes also exhibit a user's parameter to change the random seed internally used to generate random numbers to break ties etc. As discussed in [26], this parameter can randomly affect the computation and produce different branching choices and hence very different (sometime better, sometimes worse) search paths. Although the random seed does not affect the optimality of the very final solution, it can be used to diversify the MIP solver behavior so as to hopefully produce better solutions within shorter computing times. As a matter of fact, running 10 times (say) the same MIP solver on the same input data with 10 different random seeds for (say) 5 minutes, sometimes produces much better heuristic solutions than running the same MIP solver once for 50 minutes. Hence the random seed can be very useful in the heuristic mode, even more so if the runs with different seeds can be executed in parallel.

For parallel MIP codes, a main parameter is the type of parallel execution: *parallel deterministic* (meaning that the run will be reproducible due to the presence of synchronization points that however can slowdown computation), *parallel opportunistic* (generally faster as no synchronization is required), and *distributed* (meaning that the execution is distributed over a cluster of different computers).

A more advanced use of the MIP technology consists of writing a program (in any high-level programming language such as C/C++, Java, Fortran, Python, Matlab, etc.) that internally generates the model and solves it by invoking appropriate functions provided by the solver. In some cases, one can even be interested in customizing the MIP solver by exploiting some problem-specific knowledge. To this end, modern MIP solvers provide so-called *callback* functions to be invoked at the main critical points of the solution method. By default, the callbacks are not installed, meaning that they are not invoked and the solver uses its default solution strategy. By installing his/her own callbacks, an advanced user can therefore take control of the solution algorithm and customize it. We will next describe the most-used callback functions, that refer to a generic MIP solver based on the branch-and-cut solution scheme.

The *informative* callback is invoked by the solver in several parts of the code to allow the user to retrieve (e.g. to take statistics or to print) information like the value of the current-best solution (incumbent), computing time spent so far, number of branching nodes and cuts generated, etc.

The *lazy cut* callback is invoked whenever a feasible (integer) solution of the current MIP model is going to update the incumbent: this is a sort of last checkpoint where a user can discard a solution because it violates some conditions that are not explicitly part of the model. This mechanism turns out to be very useful when the problem to be solved involves an exceedingly large number of (complex) constraints, and one prefers not to include all of them explicitly in the initial model for the sake of easing its solution. If the solution passed to the lazy cut callback is not accepted for whatever reason, its infeasibility must be certified by one or more violated cuts that are automatically added ‘on the fly’ to the current model. In this way, the MIP model is enriched during the run, and only the ‘relevant constraints’ for the instance at hand are discovered and inserted explicitly in the model.

The *heuristic* callback allows a user to write a problem-specific heuristic, possibly based on the current LP solution (which is made available on input to the callback itself).

The *user cut* callback is invoked at every node just before branching, and can generate one or more cuts to be added (on the fly) to the current model with the aim of excluding the current optimal LP solution.

Additional callbacks are used to change the way the LP relaxation is solved at each node (*solve* callback), the choice of the branching variable (*branch* callback) or of the next node to process (*node* callback), etc.

## References

[1] Achterberg T, Berthold T (2007) Improving the feasibility pump. Dis Opt 4:77-86

[2] Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job shop scheduling. Mgmt Sc 34:391-401

[3] Applegate DL, Bixby RE, Chvátal V, Cook WJ (2006) The Traveling Salesman Problem. Princeton University Press.

[4] Balas E (1965) An additive algorithm for solving linear programs with 0-1 variables. Oper Res 13:517-546

[5] Balas E (1971) Intersection cuts – A new type of cutting planes for integer programming. Oper Res 19:19-39

[6] Balas E (1979) Disjunctive programming. Ann Discret Math 5:3-51

[7] Balas E (1998) Disjunctive programming: Properties of the convex hull of feasible points. Invited paper in Dis App Math 89:1-44. Originally MSRR 348, Carnegie-Mellon University, July 1974.

[8] Balas E, Carrera MC (1996) A dynamic subgradient-based branch and bound procedure for set covering. Op Res 44:875-890

[9] Balas E, Ceria. S, Cornuéjols G (1993) A lift-and-project cutting plane algorithm for mixed 0-1 programs. Math Program 58:295-324

[10] Balas E, Ceria. S, Cornuéjols G (1996) Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. Mgmt Sci 42:1229-1246

[11] Balas E, Ceria S, Cornuéjols G, Natraj N. (1996) Gomory cuts revisited. Op Res Letters 19:1-9

[12] Balas, E, Perregaard, M. (2003) A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming. Math Prog B 94:221-245

[13] Bertacco L, Fischetti M, Lodi A (2007) A feasibility pump heuristic for general mixed-integer programs. Dis Opt 4:63-76

[14] Bixby R (2013) Computational mixed integer programming. Seminar on Algorithm Engineering, Dagstuhl, Germany, September 22-27.

[15] Bixby R, Achterberg T, Rothberg E, Gu Z (2008) Recent advances in computational linear and mixed integer programming. INFORMS Optimization Society Meeting, Atlanta, March 2008.

[16] Boland NL, Eberhard AC, Engineer FG, Fischetti M, Savelsbergh MWP, Tsoukalas A (2014) Boosting the feasibility pump. To appear in Math Prog Comp, 2014

[17] Caprara A, Fischetti M, Toth P (1996) A heuristic algorithm for the set covering problem, IPCO 1996, LNCS 1084:72-84

[18] Chvátal V (1973) Edmonds polytopes and a hierarchy of combinatorial problems. Discret Math 4:305-337

[19] Cook SA (1971) The complexity of theorem-proving procedures. Proceedings of the third annual ACM Symposium on the Theory of Computing, ACM Press, New York, 151-158

[20] Cook WJ, Cunningham WH, Pulleyblank WR, Schrijver A (1998) Combinatorial Optimization. Wiley, New York

[21] Danna E, Rothberg E, Le Pape C (2005) Exploring relaxation induced neighborhoods to improve MIP solutions. Math Prog 102:71-90

[22] Edmonds J (1965) Maximum matching and a polyhedron with 0-1 vertices. J Res Nat Bureau Standards 69B:125-130

[23] Fleuren H, Goossens C, Hendriks M, Lombard M-C, Meuffels I, Poppelaars J (2013) Supply Chain-Wide Optimization at TNT Express. Interfaces 43(1):5-20

[24] Fischetti M, Glover F, Lodi A (2005) The feasibility pump. Math Prog 104:91-104

[25] Fischetti M, Lodi A (2003) Local Branching. Math Prog 98:23-47

[26] Fischetti M, Monaci M (2014) Exploiting Erraticism in Search. Op Res 62:114-122

[27] Fischetti M, Salvagnin D (2009) Feasibility pump 2.0. Math Prog Comp 1:201-222

[28] Glover F (1990) Tabu Search: A Tutorial. Interfaces 20:74-94

[29] Gomory R (1963) An algorithm for integer solutions to linear programs. In: Graves R, Wolfe P (eds) Recent Advances in Mathematical Programming. McGraw-Hill, New York, 269-302

[30] Gomory R, Johnson E (1972) Some continuous functions related to corner polyhedra. Math Prog 3:28-85

[31] Helsgaun K (2000) An effective implementation of the Lin-Kernighan traveling salesman heuristic. EJOR 126:106-130

[32] Karp R (1972) Reducibility among combinatorial problems. In R. Miller and J. Thatcher (eds), Complexity of Computer Computations. Plenum Press, New York, 85-103

[33] Koch T, Achterberg T, Andersen E, Bastert O, Berthold T, Bixby RE, Danna E, Gamrath G, Gleixner AM, Ambros M, Heinz S, Lodi A, Mittelmann H, Ralphs T, Salvagnin D, Steffy DE, Wolter, K (2011) MIPLIB 2010. Math Prog Comp 3:103-163

[34] Kroon L, Huisman D, Abbink E, Fioole P-J, Fischetti M, Maróti G, Schrijver A, Steenbeek A, Ybema R (2009) The New Dutch Timetable: The OR Revolution. Interfaces 39(1):6-17.

[35] Land AH, Doig AG (1960) An automatic method for solving discrete programming problems. Econometrica 28:497-520

[36] Lin S, Kernighan BW (1973) An effective heuristice algorithm for the traveling salesman problem. Op Res 21:498-516

[37] Lovász L, Schrijver A (1991) Cones of matrices and set functions and 0-1 optimization. SIAM J Opt, 1:166-190

[38] Nemhauser GL (2013) Integer programming: The global impact. EURO-INFORMS conference, Rome, July 2013.

[39] Nemhauser GL, Wolsey L (1988) Integer and combinatorial optimization. Wiley, New York

[40] Padberg M, Rinaldi G (1991) A branch-and-cut algorithm for the resolution of large-scale symmetric travelling salesman problems. SIAM Review 33:60-100

[41] Pinedo M, Chao X (1999) Operations Scheduling with Applications in Manufacturing and Services, McGraw-Hill

[42] Schrijver A (1986) Theory of linear and integer programming. Wiley, New York

[43] Shaw P (1998) Using programming and local search methods to solve vehicle routing problems. In Maher M, Puget JF (eds.) Principles and Practice of Constraint Programming, CP98. LNCS 1520, Springer-Verlag, 417-431.

[44] Sherali H, Adams W (1990) A hierarchy of relaxations between the continuous and convex hull presentations for 0-1 programming problems. SIAM J Dis Math 3:411-430

[45] Wolsey L (1998) Integer programming. Wiley, New York