MILP SOFTWARE

JEFFREY T. LINDEROTH
Department of Industrial and Systems Engineering, University of Wisconsin-Madison, Madison, Wisconsin

ANDREA LODI
DEIS, Universitá di Bologna, Bologna, Italy

INTRODUCTION

This article discusses software for solving a general *mixed-integer linear program* (MILP) in the form

$$\min\{c^{\mathrm{T}}x : Ax \geq b, x \geq 0,\ x_j \in \mathbb{Z}\ \forall\, j \in I\}. \quad (1)$$

The algorithmic approach relies on the iterative solution, through general-purpose techniques, of the *linear programming* (LP) relaxation

$$\min\{c^{\mathrm{T}}x : Ax \geq b, x \geq 0\}, \quad (2)$$

that is, the same as problem (1) above but the integrality requirement on the x variables in the set I has been dropped. We denote an optimal solution of problem (2) as x^*. The reason for dropping such constraints is that MILP is $\mathbb{NP}$-hard while LP is polynomially solvable and general-purpose techniques for its solution are efficient in practice.

The articles titled ***Simplex-Based LP Solvers*** and ***Interior-Point Linear Programming Solvers*** cover state-of-the-art LP techniques and solvers, while in the present article we concentrate on the basic characteristics and components of current—commercial and noncommercial—MILP solvers.

Roughly speaking, using the LP computation as a tool, MILP solvers integrate the *branch-and-bound* and the *cutting-plane* algorithms through variations of the general *branch-and-cut* scheme proposed by Padberg and Rinaldi [1] in the context of the *traveling salesman problem* (TSP). The articles titled ***Mathematical Programming Approaches to the Traveling Salesman Problem*** and ***Branch and Cut*** discuss, in detail, approaches for the TSP and the branch-and-cut approach in its full generality, respectively. Nevertheless, with the aim of giving a quick overview and fixing the notation, we briefly discuss in the following both the branch-and-bound and the cutting-plane algorithms.

The Branch-and-Bound Algorithm

In its basic version the branch-and-bound algorithm [2] iteratively partitions the solution space into sub-MILPs (the children nodes), which have the same theoretical complexity of the originating MILP (the father node or the root node if it is the initial MILP). Usually, for MILP solvers, the branching creates two children by rounding of the solution of the LP relaxation value of a fractional variable, say x_j, constrained to be integral

$$x_j \leq \lfloor x_j^* \rfloor \quad \bigvee \quad x_j \geq \lfloor x_j^* \rfloor + 1. \quad (3)$$

The two children above are often referred to as *left* (or "down") branch and *right* (or "up") branch, respectively. On each of the sub-MILPs the integrality requirement on the variables $x_j, \forall j \in I$ is relaxed and the LP relaxation is solved. Despite the theoretical complexity, the sub-MILPs become smaller and smaller because of the partition mechanism (basically some of the decisions are taken) and eventually the LP relaxation is directly integral for all variables in I. In addition, the LP relaxation is solved at every node to decide if the node itself is worthwhile to be further partitioned: if the LP relaxation value is already not smaller than the best feasible solution encountered so far, called

Wiley Encyclopedia of Operations Research and Management Science, edited by James J. Cochran

incumbent, the node can safely be fathomed because none of its children will yield a better solution than the incumbent. Finally, a node is also fathomed if its LP relaxation is infeasible.

The Cutting-Plane Algorithm

Any MILP can be solved without branching by simply finding its "right" LP description; more precisely, the *convex hull* of its (mixed-)integer solutions [3]. In order to do that, one has to iteratively solve the so-called *separation problem*

> Given a feasible solution x^* of the LP relaxation (2) which is not feasible for the MILP (1), find a linear inequality $\alpha^T x \geq \alpha_0$ which is valid for system (1), that is, satisfied by all feasible solutions $\overline{x}$ of the system (1), while it is violated by x^*, that is, $\alpha^T x^* < \alpha_0$.

Any inequality solving the separation problem is called a *cutting plane* (or a *cut*, for short) and has the effect of tightening the LP relaxation to better approximate the convex hull.

Gomory [3] has given an algorithm that converges in a finite number of iterations for pure *integer linear programming* (ILP)[1]. with integer data. Such an algorithm solves the separation problem above in an efficient and elegant manner in the special case in which x^* is an optimal basis of the LP relaxation. No algorithm of this kind is known for MILPs.

The idea behind integrating the two algorithms above is that LP relaxations (2) do not naturally well approximate, in general, the convex hull of (mixed-)integer solutions of MILPs (1). Thus, some extra work to devise a better approximation by tightening any relaxation with additional linear inequalities (cutting planes) increases the chances that fewer nodes in the search tree are needed. On the other hand, pure cutting-plane algorithms show, in general, a slow convergence and the addition of too many cuts can lead to very large LPs, which in turn present numerical difficulties for the solvers. The branch-and-cut algorithm has been proven to be very effective initially for combinatorial optimization problems (like TSP) with special-purpose cuts based on a polyhedral analysis and later on in the general MILP context.

This article is split into two parts. In the section titled "Basic Components of MILP Solvers," we discuss the basic components of nowadays MILP solvers. In the section titled "MILP Software," we list the available, commercial and noncommercial, solvers with special emphasis to their flexibility in terms of being usable within different modeling and development environments.

Some of the solvers that are discussed in the above-mentioned section have their own modeling environment and modeling language. Moreover, a recent trend for MILP software producers has been adding additional capabilities to the solvers, like that of solving some special classes of (mixed-integer) nonlinear programs (MI)NLPs. Although modeling languages and the possibility of solving MINLPs within an MILP enumerative framework are very interesting topics, they are not discussed in detail in this article. The reader is referred to the article titled for modeling environments and languages and to the articles ***Conic Optimization Software***; ***Software for Nonlinearly Constrained Optimization***; and ***MINLP Solver Software*** for NLP and MINLP software.

BASIC COMPONENTS OF MILP SOLVERS

Nowadays MILP solvers incorporate key ideas developed during the first 50 years of integer programming. In the next sections, we discuss the main ingredients in a concise way. For more details, the reader is referred to Achterberg [4] and Lodi [5] and to many articles in this encyclopedia.

Presolving

In the presolving (often called *preprocessing*) phase, the solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process.

[1]ILPs are the special case of MILPs where all variables belong to I, that is, are constrained to be integers.

This is generally done without "changing" the set of optimal solutions of the problem at hand[2] and it affects two main situations.

On the one side, it is often the case that MILP models can be improved with respect to their given formulation[3] by either removing redundant information (variables or constraints) or strengthening the variable bounds generally by exploiting the integrality of variables in I. Modern MILP solvers have the capability of cleaning up the models so as to create a presolved instance associated with the original one on which the MILP technology is then applied. The advantage is twofold: first the LP relaxation becomes smaller (then, generally, quicker to solve), and second such relaxations become stronger, that is, better approximating the convex hull of (mixed-)integer solutions. Finally, more sophisticated presolve mechanisms are also able to discover important implications and substructures that might be of fundamental importance later on in the computation for both branching purposes and cutting-plane generation.

For a detailed overview of presolving techniques, the reader is referred to Savelsbergh [7] and Martin [8].

Cutting-Plane Generation

The importance of cutting planes has been already pointed out in the introduction: without iteratively strengthening the approximation of the convex hull of (mixed-)integer solutions provided by the LP relaxation, current enumerative schemes would fail in solving difficult MILP instances.

The arsenal of separation algorithms has been continuously enlarged over the years. A large group of cuts, with strong relationship among each other, includes Chvátal–Gomory cuts, Gomory mixed-integer cuts, mixed-integer rounding cuts, $\{0, \frac{1}{2}\}$ cuts, lift-and-project cuts, and split cuts. Essentially, all these inequalities are obtained by applying a disjunctive argument, exploiting the integrality, on an integer or mixed-integer set given by a single constraint generally derived by aggregating many others. This group of cuts is presented in a brilliant and unified way by Cornuéjols [9].

Among the more "combinatorial" cutting planes, that is, those whose derivation is not directly associated with disjunctions on the integer variables, certainly the most used and useful ones are clique and cover inequalities. Cover inequalities played a fundamental role in pioneering MILP solvers [10,11].

These classes of inequalities are discussed in the sections titled "Cutting Planes" and "Automatic Convexification" in this encyclopedia.

Sophisticated Branching Strategies

The branching mechanism introduced in the introduction section requires to take two independent and important decisions at any step: node and variable selections.

Node Selection. This is very classical: one extreme is the so-called *best-bound first* strategy in which one always considers the most promising node, that is, the one with the smallest LP value, while the other extreme is *depth first* where one goes deeper in the tree and starts backtracking only once a node is fathomed, that is, it is either (mixed-)integer feasible, or LP infeasible or it has a lower bound not better (smaller) than the incumbent. The pros and cons of each strategy are well known: the former explores a less number of nodes but generally maintains a larger tree in terms of memory while the latter might need to explore a very large number of nodes. Modern software packages, typically, by default employ a hybrid of these two search techniques.

Variable Selection. The variable selection problem is the one of the methods to decide how to partition the current node, that is, on which variable to branch on in order to create the two children. For a long time, a

[2]In fact, the set of optimal solutions might change due to presolve, for example, in the case of symmetry breaking reductions; see Margot [6] and the article titled ***Heuristics in Mixed Integer Programming***.

[3]This is especially true for models originated from real-world applications and created by using modeling languages.

classical choice has been branching on the most fractional variable, that is, in the 0–1 case the closest to 0.5. This rule has been computationally shown [12] to be worse than a random selection. However, it is of course cheap to evaluate. In order to devise stronger criteria one has to do much more work. The extreme is the so-called *strong branching* technique [13], in which at any node one has to tentatively branch on each fractional variable and select the one on which the increase in the bound on the left branch times the one on the right branch is maximum. In such a version strong branching is clearly unpractical but its computational effort can be limited by (i) defining a small candidate set of variables to branch on and (ii) heuristically solving each LP by limiting *a priori* the number of simplex pivots to be performed. Other sophisticated techniques are *pseudocost branching* [14] and, the recently introduced *reliability branching* [12].

Primal Heuristics

Although primal heuristics have been part of the arsenal of the MILP solvers, almost since the beginning, only recently have they become a crucial component. This is likely in response to user demands. Solving an MILP to optimality might be less important in application than providing "good" solutions within short computing times.

The article titled ***Heuristics in Mixed Integer Programming*** is devoted to heuristics for mixed-integer programming (MIP), and two main classes of primal heuristics are considered.

On the one hand, at the root node and often at the internal nodes of the search tree, the solvers apply heuristics with the aim of converting a solution of the LP relaxation into an integer feasible solution. This is obtained through *rounding* an LP solution either component by component (without any backtracking or additional LP solve) or by *diving*, that is, rounding one or more variables and solving the LP relaxation again, iteratively. The most successful algorithm falling in this category is the *feasibility pump* heuristic proposed (for 0–1 MILPs) by Fischetti *et al*. [15].

On the other hand, once one or more feasible solutions are available, *improvement* heuristics are executed to improve the quality of the current incumbent solution. These heuristics are usually local search methods and we have recently seen a variety of algorithms which solve sub-MILPs for exploring the neighborhood of the incumbent or of a set of solutions. When these sub-MILPs are solved in a general-purpose manner, that is, through a nested call to an MILP solver, this is referred to as *MIPping* [16]. The most successful algorithms falling in this category are *local branching* by Fischetti and Lodi [17] and *relaxation induced neighborhood search* by Danna *et al*. [18].

Parallel Implementation

The branch-and-bound algorithm is a natural one to parallelize, as nodes of the search tree may be independently processed. A long body of research and implementations of parallel branch-and-bound are outlined in the survey [19]. The two types of parallel integer programming research can be loosely categorized based on the type of parallel computing architecture used. Distributed-memory architectures rely on message passing to communicate results of the algorithm. Shared-memory computers communicate information among CPUs by reading from and writing to a common memory pool.

Eckstein [20,21] was among the first to write an industrial strength parallel branch-and-bound code for general MILP. Over the past few years, there has been an emergence of *multicore* computer architectures, such that nearly all modern CPUs contain more than one unit on which computation may be performed. MILP software has followed this trend, so that many of the below surveyed packages have the ability to run multiple threads of computation, thus making use of multiple cores. For many applications, an important characteristic of the branch-and-bound algorithm is to be able to produce solutions that are *reproducible*. In parallel branch-and-bound, it is well known that the *order* in which node computations are completed can have a significant impact on performance, and often

lead to anomalous behavior [22]. An (often) undesirable consequence of this is that users can run the same instance, with the same parameter settings, and achieve very different results in terms of nodes evaluated and CPU time. To combat this undesirable behavior, modern (shared-memory-based) MILP software has introduced appropriate synchronization points in the algorithm to ensure reproducible behavior in a parallel environment. Some overhead is introduced by these synchronization mechanisms.

MILP SOFTWARE

In this section we concentrate on MILP software by first presenting a historical perspective and briefly discussing the issue of the user interface. Then, we review the main characteristics of both commercial and noncommercial MILP software packages.

Historical Perspective

The earliest commercial grade implementations of branch-and-bound algorithms for solving MILP were done by Beale and Small [23]. For a historical perspective of the development and features of early MILP software packages, such as UMPIRE, SCICONIC, and MPSX/370, the reader is referred to the article of Forrest and Tomlin [24]. These early MILP software packages developed many effective branching and node selection techniques, some of which are still in use today. A major step forward in MILP solution technology occurred in the works of Crowder *et al*. [10] and Van Roy and Wolsey [11]. These papers introduced many of the preprocessing- and structure-specific cutting-plane techniques in use today. Another important work from a computational perspective was by Balas *et al*. [25], which demonstrated that the inequalities of Gomory [26] could be effectively used as cutting planes in a branch-and-cut procedure.

Modern MILP codes synthesized many of the ideas present in the literature, taking the best points of these works and engineering them into commercial-grade software packages. The article of Bixby and Rothberg [27] offers a nice historical exposition of how MILP software improved by orders of magnitude over a period of a decade, starting in the late 1990s.

User Interfaces

An important aspect of the design of software for solving MILPs is the user interface. The range of purposes for MILP software is quite large, so it stands to reason that the number of user interface types is also large. In general, users may wish to solve MILPs using the solver as a "black box," they may wish to call the software from a third party package, or they may wish to embed the solver into custom applications, which would require software to have a callable library. Often, the user may wish to adapt certain aspects of the algorithm. For instance, the user may wish to provide a custom branching rule or problem-specific valid inequalities. The customization is generally accomplished either through the use of language callback functions or through an abstract interface wherein the user must derive certain base classes and override default implementations for the desired functions. Nearly all of the below surveyed software packages have stand-alone executables, callable libraries, and allow the user to customize the branch-and-bound algorithm to varying degrees.

MILP Software Packages

Any review of software features is inherently limited by the temporal nature of software itself. In fact, algorithmic advances in the field of computational integer programming, many previously described in this article, can quickly render obsolete MILP software that was once state of the art. Here, we give a brief overview of what are the most widely used MILP software packages at the time of publication. Hans Mittelmann has for many years run independent benchmarks of MILP software, and been publishing the results. In general, the commercial software significantly outperforms noncommercial software on the instances in the Mittelmann suite, but it is difficult to draw strong conclusions about the relative performance of different commercial systems. Results of these benchmarks can be found

at http://plato.asu.edu/ftp/milpf.html and http://plato.asu.edu/ftp/milpc.html. Many of these solvers are available for use free of charge on the NEOS server website http://www-neos.mcs.anl.gov.

Commercial Software Packages. Unsurprisingly, exact implementation details of specific commercial MILP software packages are not known, as revealing such details would be a competitive disadvantage. In this article, information from the various software user manuals was used to compile features specific to each package. Further, it is impossible to detail all features of a software package, so we focus on those features that are, in the opinion of the authors, particularly noteworthy. Many of the commercial software packages listed here have free or limited cost licensing options available for academic users.

CPLEX

Version: 12.2

Website: http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

Interfaces: C, C++, Java, .NET, Matlab, Python, Microsoft Excel

CPLEX is a commercial MILP software package owned and distributed by IBM. The branch-and-cut algorithm contains a full suite of presolving techniques, cutting planes, search strategies, and heuristic techniques for finding feasible solutions. A special search algorithm in CPLEX, called *dynamic search*, can be used instead of traditional branch-and-cut. CPLEX also includes a `mipemphasis` metaparameter that adjusts many algorithmic parameters simultaneously to help users achieve a high-level goal. CPLEX contains facilities for automatically tuning MILP algorithm parameters for a particular class of instances. CPLEX may parallelize the branch-and-bound search using multiple threads in either a deterministic or nondeterministic manner. A final notable feature of CPLEX is its ability to enumerate multiple solutions that are close to optimality and store them in a *solution pool*.

Gurobi

Version: 3.0

Website: www.gurobi.com

Interfaces: C, C++, Java, Python, .NET, Matlab

Gurobi Optimizer contains a relatively new MILP solver that was built from scratch to exploit modern multicore processing technology. Gurobi contains all advanced algorithmic features, including a variety of cutting planes, heuristics, and search techniques. Notable features of Gurobi include an `MIPFocus` metaparameter that simultaneously controls many algorithmic parameters to achieve the desired goal. Gurobi may execute the branch-and-bound algorithm using multiple computational threads. The parallel search is done in a deterministic manner. The Gurobi interactive shell is implemented as a set of extensions to the Python shell. Gurobi is also available "on demand" using the Amazon Elastic Compute Cloud.

LINDO

Version: 6.1

Website: www.lindo.com

Interfaces: C, Visual Basic, Matlab, Ox

LINDO Systems offers an MILP solver as part of its LINDO API. The MILP solver offers 15 different types of cutting planes, 6 different node selection rules, and a variety of preprocessing techniques and branching rules.

Mosek

Version: 6.0

Website: www.mosek.com

Interfaces: C, C++, Java, .NET, Python

MOSEK ApS is a company specializing in generic mathematical optimization software, and their suite of software includes a solver for MILP. The solver has 6 node selection rules, 11 types of cutting planes, the feasibility pump, and other heuristics, as well

as advanced branching methodologies such as strong branching. Mosek is available for use, through a GAMS interface, on the NEOS server.

XPRESS-MP
Version: 7.0
Version: http://www.fico.com/en/Products/DMTools/xpress-overview/Pages/Xpress-Optimizer.aspx
Interfaces: C, C++, Java, .NET, VBA

FICO Xpress Optimization Suite 7 offers a branch-and-cut-based software for solving MILP. XPRESS-MP offers many advanced cutting planes, including an effective implementation of lift-and-project cuts [28]. Modern preprocessing, heuristic, and branching techniques are also available. A unique feature of XPRESS-MP is that it offers an option to branch into general (split) disjunctions, or to search for special structures on which to branch. XPRESS-MP has features to enumerate multiple feasible solutions, and also to tune algorithmic parameters. XPRESS-MP is available for use, with three different input formats, on the NEOS server.

Noncommercial MILP Packages. Here, we give a cursory description of the most popular noncommercial MILP software packages. The paper of Linderoth and Ralphs [29] contains a more in-depth treatment of noncommercial software packages.

BLIS
License: Common Public License
Version: 0.91
Website: https://projects.coin-or.org/CHiPPS
Language: C++

BLIS is an open-source MILP solver available as part of the COIN-OR project. It is built on top of the COIN-OR high-performance parallel search (CHiPPS) framework, so it has been designed to run on distributed-memory platforms (one of the few available MILP software packages with this feature). Linear programs are solved using the COIN-OR linear programming (Clp) solver, and cutting planes from the COIN cut generation library (CGL) are used.

CBC
License: Common Public License
Version: 2.5
Website: https://projects.coin-or.org/Cbc
Language: C++

CBC is an open-source MILP solver distributed under the COIN-OR project. It is built from many COIN components, including Clp and CGL. It is available both as a library and a stand-alone solver. CBC may be parallelized using shared-memory parallelism, and there are a large number of examples demonstrating its use. CBC is available for use on the NEOS server.

GLPK
License: GNU General Public License (GPL)
Version: 4.44
Website: http://www.gnu.org/software/glpk/
Language: C

GLPK is the GNU LP kit, a set of subroutines comprising a callable library and black box solver for MILP. The software distinguishes itself by being available under the GNU GPL and through the large number of community-built interfaces available for its use. GLPK is available for use, though a GAMS interface, on the NEOS server.

lp_solve
License: GNU lesser general public license (LGPL)
Version: 5.5
Website: http://lpsolve.sourceforge.net/5.5/
Language: C

The software lp_solve is an open-source linear and integer programming solver. There are a large number of interfaces available through which lp_solve can be used, including Java, AMPL, MATLAB, O-Matrix,

Sysquake, Scilab, Octave, Freemat, Euler, Python, Sage, PHP, and R.

MINTO
License: Given as library only
Version: 3.1
Website: http://coral.ie.lehigh.edu/minto/
Language: C

MINTO (Mixed INTeger Optimizer) is a black box solver and solver framework for MILP. MINTO is available only in library form. MINTO relies on external software to solve the LP relaxations that arise during the algorithm. Primary development of the software was done in the 1990s. The code was noteworthy at that time for a dynamic preprocessing engine [30] and effective lifted cover inequalities [31]. MINTO is available for use, through an AMPL interface, on the NEOS server.

SCIP
License: ZIB Academic License
Version: 1.2
Website: http://scip.zib.de/
Language: C

SCIP is an MILP software package developed and distributed by a team of researchers at Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB). SCIP is also a framework for constraint integer programming and branch-cut-and-price, allowing the user significant control of the algorithm. Development of SCIP was awarded the Beale Orchard Hayes Prize in 2009 [32]. Current benchmarks indicate that SCIP is likely the fastest noncommercial MILP solver. Other references for SCIP include the work of Achterberg [33] and Achterberg *et al*. [34] . SCIP is available for use, through a variety of interfaces, on the NEOS server.

SYMPHONY
License: Common Public License
Version: 5.2
Website: http://www.coin-or.org/SYMPHONY/index.htm
Language: C

SYMPHONY is a black box solver and callable library for MILPs. The core solution methodology of SYMPHONY is a customizable branch, cut, and price algorithm that can be executed sequentially or in parallel [35]. SYMPHONY calls on several other open-source libraries for specific functionality, including COIN-OR's CGL and Clp. There are several unique features of SYMPHONY that are worth mentioning. First, SYMPHONY contains implementation of an algorithm for solving bicriteria MILPs and methods for approximating the set of Pareto outcomes [36]. SYMPHONY also contains functions for local sensitivity analysis [37]. Second, SYMPHONY has the capability to warm start the branch-and-bound process from a previously calculated branch-and-bound tree, even after modifying the problem data. The work by Ralphs and Guzelsoy [38] gives an overview of the SYMPHONY callable library. SYMPHONY is available for use via MPS files on the NEOS server.

CONCLUSIONS

As decision processes become increasingly complex, the need for formal tools to aid in both planning and operation of the underlying system becomes more important. Many disciplines have embraced the use of optimization as a central tool to enhance the quality of decisions made. A large fraction of the models used in commercial settings are MILPs. Powerful, reliable, and flexible software systems (both commercial and noncommercial) are a key component of this broad-based acceptance. The field of MILP and computational aspects therein remain an active area of research. Given the high commercial penetration of MILP planning models, undoubtedly future research advances will find themselves implemented in software.

REFERENCES

1. Padberg MW, Rinaldi G. A branch and cut algorithm for the resolution of large-scale symmetric traveling salesmen problems. SIAM Rev 1991;33:60–100.

2. Land AH, Doig AG. An automatic method of solving discrete programming problems. Econometrica 1960;28:497–520.
3. Gomory RE. Outline of an algorithm for integer solutions to linear programs. Bull Am Math Soc 1958;64:275–278.
4. Achterberg T. Constraint integer programming [PhD thesis]. Berlin: ZIB; 2007.
5. Lodi A. MIP computation. In: Jünger M, Liebling TM, Naddef D, Nemhauser GL, Pulleyblank WR, Reinelt G, Rinaldi G, Wolsey LA, editors. 50 years of integer programming 1958-2008. Heidelberg: Springer; 2009. pp. 619–645.
6. Margot F. Symmetry in integer linear programming. In: Jünger M, Liebling TM, Naddef D, Nemhauser GL, Pulleyblank WR, Reinelt G, Rinaldi G, Wolsey LA, editors. 50 years of integer programming 1958-2008. Heidelberg: Springer; 2009. pp. 647–686.
7. Savelsbergh MPW. Preprocessing and probing techniques for mixed integer programming problems. ORSA J Comput 1994;6:445–454.
8. Martin A. General mixed integer programming: computational issues for branch-and-cut algorithms. In: Jünger M, Naddef D, editors. Computational combinatorial optimization. Volume 2241, Lecture notes in computer science. Berlin/Heidelberg: Springer; 2001. pp. 1–25.
9. Cornuéjols G. Valid inequalities for mixed integer linear programs. Math Program 2008;112:3–44.
10. Crowder H, Johnson E, Padberg MW. Solving large scale zero-one linear programming problem. Oper Res 1983;31:803–834.
11. Van Roy TJ, Wolsey LA. Solving mixed integer programming problems using automatic reformulation. Oper Res 1987;35:45–57.
12. Achterberg T, Koch T, Martin A. Branching roles revisited. Oper Res Lett 2005;33: 42–54.
13. Linderoth JT, Savelsbergh MWP. A computational study of search strategies for mixed integer programming. INFORMS J Comput 1999;11:173–187.
14. Benichou M, Gauthier JM, Girodet P, *et al*. Experiments in mixed-integer programming. Math Program 1971;1:76–94.
15. Fischetti M, Glover F, Lodi A. The feasibility pump. Math Program 2005;104:91–104.
16. Fischetti M, Lodi A, Salvagnin D. Just MIP it! In: Maniezzo V, Stützle T, Voss S, editors. MATHEURISTICS: hybridizing metaheuristics and mathematical programming. Operations research/computer science interfaces series. Heidelberg: Springer; 2009. pp. 39–70.
17. Fischetti M, Lodi A. Local branching. Math Program 2002;98:23–47.
18. Danna E, Rothberg E, Le Pape C. Exploiting relaxation induced neighborhoods to improve MIP solutions. Math Program 2005;102:71–90.
19. Gendron B, Crainic TG. Parallel branch and bound algorithms: survey and synthesis. Oper Res 1994;42:1042–1066.
20. Eckstein J. Parallel branch-and-bound algorithms for general mixed integer programming on the CM-5. SIAM J Optim 1994;4:794–814.
21. Eckstein J. Parallel branch-and-bound methods for mixed integer programming. SIAM News 1994;27:12–15.
22. Lai TH, Sahni S. Anomalies in parallel branch and bound algorithms. Proceedings of the 1983 International Conference on Parallel Processing; Columbus (OH). 1983. pp. 183–190.
23. Beale EML, Small RE. Mixed integer programming by a branch and bound method. In: Kalenich WH, editor. Volume 2, Proceedings IFIP Congress 65. Washinton (DC): Spartan press and London: Macmillan; 1965. pp. 450–451.
24. Forrest JJH, Tomlin JA. Branch and bound, integer and non-integer programming. Ann Oper Res 2007;149:81–87.
25. Balas E, Ceria S, Cornuéjols G, *et al*. Gomory cuts revisited. Oper Res Lett 1996;19:1–9.
26. Gomory RE. An algorithm for the mixed integer problem. Technical Report RM-2597. The Rand Corporation; 1960.
27. Bixby R, Rothberg E. Progress in computational mixed integer programming – a look back from the other side of the tipping point. Ann Oper Res 1007;149:37–41.
28. Balas E, Perregaard M. A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming. Math Program 2003;94:221–245.
29. Linderoth JT, Ralphs TK. Noncommercial software for mixed-integer linear programming. In: Karlof JK, editor. Integer programming: theory and practice. Operations research series. CRC Press; 2005. pp. 253–303.
30. Atamtürk A, Nemhauser G, Savelsbergh MWP. Conflict graphs in solving integer

programming problems. Eur J Oper Res 2000;121:40–55.

31. Gu Z, Nemhauser GL, Savelsbergh MWP. Cover inequalities for 0-1 linear programs: computation. INFORMS J Comput 1998;10:427–437.
32. Achterberg T. SCIP: solving constraint integer programs. Math Program Comput 2009;1(1): 1–41.
33. Achterberg T. Constraint integer programming [PhD thesis]. Berlin: Technischen Universtät Berlin; 2007.
34. Achterberg T, Berthold T, Koch T, *et al*. Constraint integer programming: a new approach to integrate CP and MIP. In: Perron L, Trick MA, editors. Integration of AI and OR techniques in constraint programming for combinatorial optimization problems. Volume 5015, Lecture notes in computer science. Heidelberg: Springer; 2008. pp. 6–20.
35. Ralphs TK, Ladányi L, Saltzman MJ. Parallel branch, cut, and price for large-scale discrete optimization. Math Program 2003;98:253–280.
36. Ralphs T, Saltzman M, Wiecek M. An improved algorithm for biobjective integer programming. Ann Oper Res 2006;147:43–70.
37. Schrage L, Wolsey LA. Sensitivity analysis for branch and bound linear programming. Oper Res 1985;33:1008–1023.
38. Ralphs TK, Guzelsoy M. The SYMPHONY callable library for mixed integer programming. The Proceedings of the Ninth Conference of the INFORMS Computing Society; Annapolis (MD). 2005. pp. 61–71.