Vol. 14, No. 1, September 2013, pp. 26–38
ISSN 1532-0545 (online)
INFORMS Transactions on Education
http://dx.doi.org/10.1287/ited.2013.0115

A Survey of Linear and Mixed-Integer Optimization Tutorials

Alexandra M. Newman
Division of Economics and Business, Colorado School of Mines, Golden, Colorado 80401, newman@mines.edu
Martin Weiss
European Commission, Directorate-General Joint Research Centre, Institute for Energy and Transport, Sustainable Transport Unit, I-21027 Ispra-Italy, martin.weiss@jrc.ec.europa.eu

As advanced undergraduate and graduate students in optimization begin conducting research, they must base their work on articles found in academic journals. However, there is often a gap between the levels at which a journal article and a textbook are written. Bridging that gap are tutorials on fundamental, yet advanced, concepts such as (i) algorithmic details of linear and mixed-integer optimizers, (ii) formulations that render models more tractable, (iii) descriptions of the mathematical structure of linear and mixed-integer programs, and (iv) manuals of modeling languages that enable quick implementation of formulations and of linear and mixed-integer solvers. This survey paper provides references to papers and reports whose purpose is to give overviews of linear and mixed-integer optimization. We also include seminal texts and journal articles on fundamental topics, and furnish references on applications whose implementation may have implications for more general problems with similar mathematical structure. We propose to guide graduate students to identify references that aid in their understanding of advanced journal articles and that help them write their own research articles.

Key words: teaching engineering, teaching modeling, teaching optimization, linear optimization, mixed-integer optimization, tutorials, efficient formulations, cuts, heuristics, modeling languages, decomposition techniques

1. Introduction

Students in their first year or two of college often find that consulting the class textbook suffices for solving their carefully contrived homework problems that test a small number of structured concepts, generally, those being taught in the class. However, advanced undergraduate and graduate students realize that their missions, which assume the form of, inter alia, class projects and research, involve open-ended questions, for which there is little, if any, guidance provided in standard textbooks, and only some difficult-to-parse direction given in the open academic literature.

To bridge the gap between textbook material and journal articles, students should be aware of seminal references in their field, tutorials that summarize advanced concepts, and manuals that help practitioners use software to solve their problems. In this survey, we categorize helpful references into those that treat (i) linear programming, (ii) integer programming, including constraint programming, (iii) optimization under uncertainty, (iv) solution techniques specific to problem structure, (v) modeling languages and solvers, (vi) a few applications, and (vii) seminal textbooks. Throughout this tutorial, we assume a basic knowledge of optimization terminology at the level expected of an advanced undergraduate or of a graduate student in engineering or applied mathematics.

Although topics such as complexity theory and nonlinear programming are relevant, we omit these from our treatment. For complexity theory, we recommend Tovey (2002), who (i) introduces big-O time bound notation; (ii) differentiates problems classified as P, NP, NP-complete, and NP-hard; (iii) introduces yes-no form with examples; and then (iv) presents examples of how to identify "hard" problems. Johnson (2012) gives a history of NP-completeness. For nonlinear programming, we recommend Bazaraa et al. (2006) for continuous nonlinear programming, and Lee and Leyffer (2012) for nonlinear programming with discrete variables. We also omit dynamic programming and its variants, but refer the reader to Powell (2009), who presents approximate dynamic programming as a tool for modeling and solving a broad class of complex problems that typically involve uncertainty and for which decisions must be

made over time in fields of study such as control theory (including classical engineering and economics), artificial intelligence, and operations research. Finally, we omit a treatment of network models, such as spanning tree problems (Němeček and Němečková 2012), shortest path problems (Schrijver 2012a), and maximum flow and transportation problems (Schrijver 2012b), but refer the reader to these references and to the seminal text of Ahuja et al. (1993).

Before we begin, it is important to reflect on the origins of operations research. Gass and Assad (2011) provide a description and list influential people during the early years of the field. They then describe seminal works themselves, and papers that discuss seminal works. After a description of early operations research-focused centers at locations such as the RAND Corporation, the Pentagon, MIT, and Princeton, Gass and Assad (2011) conclude with stories, lessons learned, and an annotated bibliography. Grötschel (2012) edits a lengthier volume that contains articles on classical mathematicians and their contributions to the field of optimization, as well as accounts detailing the history and significant developments in linear programming, discrete optimization, nonlinear programming, and related computational work. Where relevant and appropriate throughout our article, we give specific references to seminal papers and papers that describe the history of a subfield. However, we provide only a few such papers, and instead focus on state-of-the-art tutorials.

2. Linear Programming

Arguably, the seminal contributions to the field of inequality-constrained optimization began with Dantzig's theory and algorithms of linear programming in which both the objective function and the constraints of an optimization model are linear functions of the decision variables; these variables are allowed to assume continuous (fractional) values. We first discuss the earlier simplex method, followed by interior point methods, and cite papers that give a performance contrast between the two types of linear programming algorithms. We conclude this section by citing a paper on implementation issues for difficult-to-solve linear programs. Dantzig et al. (1955) represents a fundamental paper on the simplex method applied to both maximization and minimization problems. The authors include the theory behind the construct of a basis, the existence of a (basic) feasible solution and a bounded linear program, the existence of an optimal solution, and the finite termination of the algorithm. Lemke (1954) describes the dual simplex method, applies this method to the dual problem, discusses the advantages of this approach for solving linear programs, and concludes with some remarks regarding dual degeneracy, numerical stability, and Phase I. Despite its usefulness in practice, the simplex algorithm has exponential worst-case complexity (Klee and Minty 1972).

Based on the premise that there exists a polynomial time algorithm for linear programming (Khachian 1979), Karmarkar (1984) proposed an alternative method for solving linear programs, now called the barrier, or interior point, method. Karmarkar's seminal paper on the algorithm first gives an overview of the background for and the main idea of this polynomial-time algorithm, before providing both algorithmic and theoretical details in subsequent sections. The author also discusses alternate implementations and examples of practical performance. Simply because the theoretical complexity of the simplex method is worse than that of interior point algorithms does not mean that the simplex method always performs worse in practice. The implementations of the two types of algorithms is important for run-time performance, as is the type of linear program being solved. Gill et al. (1986) provide a contrast between the simplex method and the barrier method, review the barrier method, and describe the notion of a barrier function. The authors then equate Karmarkar's method with a "projected Newton barrier method" in the context of linear programming, and demonstrate that they can construct an interior point algorithm for linear programs based on a logarithmic transformation to inequality constraints. They present numerical results with and without scaling constraint coefficients, and contrast simplex and barrier method performance. Along similar lines, Marsten et al. (1990) motivate interior point methods based on techniques for converting equality- and inequality-constrained optimization problems to unconstrained problems, and based on using Newton's method to solve such unconstrained problems. The authors mathematically motivate the derivation of interior point methods, and discuss implementation issues and computational results, including contrasts with simplex method performance.

Two survey papers coalesce the improvements in interior point algorithm performance since a variant of the algorithm was introduced in the 1980s. Lustig et al. (1994) provide a state-of-the-art treatment of interior point algorithms about a decade after their emergence. Specifically, the authors examine various implementations of logarithmic barrier functions (that penalize constraint violation in the objective, thus inducing feasibility), and matrix factorization methods, among others, and show computationally that carefully constructed interior point implementations can outperform simplex method implementations on large-scale problems. Terlaky (2009) reviews interior point methods, their variants and their benefits, and

develops theory in support of a basic interior point method. The author then discusses limitations of interior point methods in terms of their ability to handle problems with many constraints or problems that are ill conditioned or numerically unstable. Terlaky (2009) concludes with open problems regarding theoretical performance, and with extensions of interior point methods beyond applications in linear optimization.

In an advanced paper, Murty (2009) describes sphere methods for solving large-scale, and not necessarily sparse, linear programs. The author briefly explains the importance of Gaussian elimination, the simplex method, and interior point methods for solving linear systems. He then defines sphere methods as special cases of interior point methods in which few, or no, matrix inversions are required, and the matrix inversion that is required involves only a small subset of all constraints. Based on the premise that the primal-dual interior point method is the most popular interior point variant because of its termination criterion and the availability of both primal and dual solutions, the author shows how to transform sphere algorithms into primal-dual algorithms. The method assumes the possession of an initial feasible solution, uses centering (based on a ball), and determines a descent direction to iterate from one solution to the next. The author concludes with numerical results and recommendations for further enhancements of the method.

However, these papers do not diminish the prominence of the simplex method as the algorithm of choice for solving practical linear programming problems. In fact, most popular commercial linear programming solvers (see §7) incorporate the primal and dual simplex methods, as well as one or more interior point methods. Such software allows the user to select the solver and to tune it by choosing, for example, pricing schemes in the simplex method, or features to emphasize scaling of the problem instance. Klotz and Newman (2013a) describe suggestions for improving computational performance of these solvers for difficult linear programs. Their suggestions include (i) consideration of parameter settings applied to a problem instance, (ii) algorithm selection based on the mathematical structure of the problem, and (iii) assessment of the numerical stability associated with the problem instance. Solver output from an iteration log serves as guidance.

3. Integer Programming

A generalization of linear programming is one in which the objective function and the constraints of an optimization model remain linear functions of the decision variables, but some or all of these variables assume discrete or integer values. In this section, we provide references that give background on the subject, discuss the history of computational advancements, describe ways in which model tractability can be enhanced and solutions can be improved, provide descriptions of integer programming enhancements in two specific cases, and conclude with contemporary advice on computation. Cook (2012) describes a basic method for solving integer programming problems, the branch-and-bound algorithm, and provides background on the fundamentals that are connected to develop it. He gives the origin of the algorithm's name, and concludes with an extension to the branch-and-cut algorithm. His article points out the early recognition of the necessity for modeling practical problems using integer variables. Logical constructs, among others, require such variables, but during the 1950s, even small linear programming problems were not tractable. However, several decades later, there had been significant hardware and software developments regarding the implementation of algorithms for solving integer programs. Hoffman (2000) describes the improvements in tractability of large-scale mixed-integer linear optimization models from the beginning to the end of the 1990s. She provides examples of good formulation and discusses (i) exact solution strategies including enumeration, decomposition methods (see §6 of this paper), and cutting plane algorithms; and (ii) heuristic techniques, including simulated annealing and tabu search. She then describes column generation (see §6 of this paper), and hybrid algorithms, the latter of which combine some or all of the aforementioned techniques. She discusses parallel implementation and concludes with new directions for modeling and solution examination, and for modeling under uncertainty (see §5 of this paper).

These advances in the ability to solve realistically sized integer programming instances are explained and documented in a pair of papers (Bixby et al. 2000, Bixby and Rothberg 2007). Whereas the earlier paper concentrates to some extent on linear programs and then discusses the improvements in integer programming tractability owing to heuristics, the presolve feature, and cutting planes, the latter paper reviews more recent progress in solving mixed-integer programming problems; this progress comes both in terms of hardware capabilities and in terms of software (specifically, CPLEX (IBM 2009)) sophistication. The authors run the branch-and-bound algorithm, disabling then-recent features, most notably, cutting planes, to demonstrate their effectiveness. Bixby and Rothberg (2007) conclude by emphasizing the usefulness of the callback feature in CPLEX, which allows users to implement tailored techniques within the optimization algorithm.

A number of authors recognize the extent to which model formulation affects tractability, and administer advice to this end. Camm et al. (1990) provide

specific formulation guidance for practitioners interested in the ubiquitous big "*M*." This "*M*" stands in place of a very large number, and is often used as a coefficient to represent the ability to employ "as much of the corresponding resource as one desires" when the binary variable linked to incurring a fixed charge for such use has a value of 1. (Otherwise, one cannot use any of the resource.) The authors mention many practical examples to demonstrate that almost always, there is a maximum practical size for "*M*." Furthermore, they illustrate that using unnecessarily large values for this parameter diminishes model tractability. Trick (1997) presents an unmaintained, but still highly relevant, website containing examples, and discussions of and formulations for popular integer programming problems. Brown and Dell (2007) provide excellent examples of how to start formulating a model, focusing on commonly appearing integer linear programming structure, for example, (i) the use of big "*M*"; (ii) packing, covering, and partitioning constraints; (iii) cardinality constraints; and (iv) the use of the inclusive and exclusive "or." The authors begin the tutorial without set notation but then provide examples using set notation. They draw on examples from the military, and conclude their paper by specifying an easy-to-read format in which mathematical programming models should be written. Trick (2005) points out that some well-known reformulation techniques designed to improve the linear programming relaxation bound for an integer program do not necessarily apply in light of more sophisticated integer programming solvers, which already recognize common tightening inequalities. He suggests instead, via two examples in transportation and sports scheduling, that redundant constraints (which are relaxations of existing constraints), and variable redefinition can significantly improve solution times of model instances.

Preprocessing of an already-formulated model before it is passed to the branch-and-bound algorithm can improve the tractability of an integer program significantly. Some preprocessing techniques are embedded in solvers, and should serve as a secondary screening mechanism on a well-formulated model. (However, preprocessing can also be applied to a poorly formulated model.) Savelsbergh (1994) identifies such techniques to detect redundancies and infeasibility, and to tighten variable bounds and alter coefficients in integer programming problems. He also determines opportunities to fix (binary) variable values based on infeasibilities associated with having the variable assume alternate values within its domain. Additionally, objective function coefficients can be modified by comparing these coefficients to those on the same variables in the constraint set. The author then extends the techniques, and provides numerical examples and computational results.

In a pair of eminently readable papers, authors suggest how to improve the quality of solutions obtained from integer programs. Sherali and Smith (2001) reduce solution times of integer programming formulations possessing symmetry, or many solution options with similar mathematical characteristics. The authors present three examples: network design, noise pollution reduction, and machine scheduling; they show how, in each case, careful formulation that provides a hierarchy in solutions can lead to a more tractable model. In some cases, a model can also be reformulated to achieve better results. Brown et al. (1997) explain the concept of "persistence," or the art of formulating models that yield similar solutions with a change in some input parameters. While this concept can be applicable to increasing model tractability, it also pertains to formulating models well under repeated solves, where the repetition is introduced because of changing conditions (inputs). Implementing the ideas of Brown et al. (1997) helps to mitigate the commonly occurring characteristic of integer programs that a small change in data can yield a vastly different solution (where the difference in solutions is measured, e.g., for binary solutions, in terms of the Hamming distance between them). In practice, one is often willing to give up some value (or cost) in the objective function for a solution similar to one obtained with slightly different inputs. For example, a persistent solution might be welcome in a setting in which a predetermined production schedule must be reoptimized because a machine is inoperable.

Some authors provide advice pertaining to more specific classes of integer programming problems. Rebennack et al. (2012) address the maximum stable set problem, i.e., maximizing the sum of the weights in a set of nodes on a graph such that the nodes in the set are pairwise nonadjacent. The authors discuss special constructs and specific solution strategies, e.g., preprocessing heuristics, and a branch-and-cut algorithm. Cornuéjols (2008) discusses families of cuts that strengthen the linear programming bound of an integer program. His tutorial reviews basic polyhedral theory, and lift-and-project theory. The author then derives mixed-integer inequalities, e.g., mixed-integer rounding cuts and Gomory mixed-integer cuts, and presents examples and computational results for several classes of these types of inequalities. The conclusions contain a schematic of the relationship between different classes of cuts, and avenues for future research. Cornuéjols (2012) gives a more in-depth discussion that focuses only on Gomory cuts, including their checkered past regarding the extent to which they were thought to be useful.

Similar to linear programs, good commercial solvers for integer programs also exist (see §7). However, for maximal performance, one must be familiar not only with good formulation but also with the attributes of the solver. Danna (2008) discusses the performance variability of integer programs, defined as an inexplicable difference in computation time on the same model instance under different performance-neutral circumstances, e.g., a similar algorithm in a similar environment. She includes in her presentation the definition of performance variability, its causes and effects, and suggests ways in which to reduce variability. Klotz and Newman (2013b) describe methods for improving computational performance of difficult mixed-integer linear programs. They suggest tuning algorithmic parameter settings, implementing cuts, defining variables effectively, formulating the model well, and being able to identify, based on the mathematical structure of the problem and/or the output from a node log, when to employ which technique(s).

4. Constraint Programming

Constraint programming is a logic-based method for examining and solving integer programming problems which usually lack an objective function, or that possess only a simple objective. With its roots in computer science, its techniques reduce the search space by eliminating infeasible solutions from consideration using constraint satisfiability arguments. Optimization problems involving binary or integer variables, and ones in which finding a feasible solution can be difficult, e.g., scheduling and timetabling, lend themselves well to this technique. This section highlights several papers that describe the method of constraint programming, and contrast it with classical mathematical (integer) programming. Hooker (1994) examines pure and mixed-integer programs. He draws analogies between cutting planes and other classical methods designed to expedite solutions within a traditional branch-and-bound framework, and describes reasoning used to determine feasible solutions via constraint and logic-programming. The author emphasizes that constraint programming can exploit the structure of some (mixed) integer programs, and rejects the idea that the inability to convert inequality constraints into a logical form precludes the effectiveness of constraint programming. Hooker (1994) provides examples of how logical implications can be derived from an integer programming problem, and presents a generic branch-and-bound algorithm with in-built logic processing. He also mentions relaxations, strong cuts, and nonvalid cuts within the context of branch-and-bound and constraint programming.

Smith (1995) provides an introduction to constraint satisfaction problems, including the domain over which variables are generally defined, the types of constraints often present in these problems, and algorithms. The author also discusses ordering techniques to expedite solutions, and mentions extensions involving an objective function. Brailsford et al. (1999) provide a tutorial on constraint satisfaction problems. The authors define these combinatorial problems as those requiring a solution that satisfies a set of constraints. The authors contend that traditional integer programming techniques do not necessarily perform well on these problems, and suggest that artificial intelligence researchers have introduced efficient algorithms that are relatively unknown to operations researchers. The authors give the basic components of a constraint satisfaction problem, provide examples, describe exact algorithms, suggest efficient formulations (e.g., ones in which symmetry is precluded and the number of variables per constraint is small), and introduce heuristics.

In a rather accessible paper, Lustig and Puget (2001) contrast constraint programming and classical mathematical programming. The authors then provide three examples: a graph coloring problem, the stable marriage problem, and a sequencing problem, and discuss how the constraint programming concepts of domain reduction and constraint propagation can be used to apply search strategies to solve these classical mathematical programming problems as constraint programs. Suggestions include hybrid strategies that combine mathematical programming and constraint programming to enhance tractability, and examples of warehouse location-allocation and crew scheduling conclude the paper.

5. Optimization Under Uncertainty

Researchers have long been aware of the potential shortcomings of treating input data as deterministic. Most of the work cited in this section addresses uncertain optimization in the context of linear programming. We first cite articles that provide background on the topic, and then discuss proper stochastic programming model formulation, and two specific types of stochastic programs. We conclude with one paper on simulation optimization and another on robust optimization.

Dantzig (1955) and Beale (1955) independently discovered an extension of linear programming that handles uncertainty. Dantzig's paper discusses two-stage stochastic linear programs in which first-stage decision variables must be determined before observing some yet-to-be-realized scenarios, whereas second-stage decision variables can be selected after observing these scenarios; hence, the values of the second-stage variables depend on the scenarios. Dantzig further considers the multistage extension of this two-stage

paradigm. Beale focuses on the mathematical structure of such models, emphasizing cases in which convexity properties arise. He considers a formulation in which the optimal value of the second-stage problem is a convex function in the first-stage decision vector and, hence, the expected value of the second-stage cost is also a convex function of these variables (because it is a positively weighted sum of convex functions). He further provides conditions under which the optimal value of the second-stage problem is a convex function with respect to the uncertain parameters in the constraint set.

Sen and Higle (1999) provide guidance in formulating simple recourse, two-stage, and multistage stochastic linear programs. They begin by postulating that deterministic linear programs that use expected values for the data yield quantitatively and qualitatively different solutions than their stochastic programming counterparts. The authors contrast solutions obtained from expected-value data with those from wait-and-see (or scenario) analysis, and introduce examples to illustrate possible characteristics of stochastic programming, including future infeasibility, and nonconvex feasible regions. They conclude with scenario analysis and recent developments to simplify stochastic programs. The tutorial on the same subject by Higle (2005) shows an example in which sensitivity analysis is not sufficient to examine an uncertain scenario, and compares the outcome from this analysis to one in which recourse is used. She then provides a general introduction to the concept of recourse, and the ideas of two-stage, simple, fixed, complete, and multistage recourse. She discusses solutions for these types of problems, demonstrates using an example why a stochastic program solution can be better than a corresponding deterministic model solution, discusses solution techniques, and concludes with computational results and further reading.

Rockafellar (2007) discusses ways in which to address risk when optimizing a system under uncertainty, where the uncertainty is characterized as a set of scenarios, each of which has a given nonzero probability of occurring. He discusses traditional approaches to optimization modeling under uncertainty, including guessing which scenario might materialize and optimizing with respect to it, performing worst-case analysis, and using expectations and standard deviations. The author then describes means to hedge against uncertainty by using techniques that lend themselves to multistage modeling, e.g., stochastic programming and dynamic programming. Rockafellar then provides means for quantifying risk in a coherent fashion, describes the measures of value-at-risk and conditional value-at-risk, and concludes with examples, generalizations, and characterizations of what an optimal solution under uncertainty is.

Ahmed and Shapiro (2008) address a chance-constrained stochastic program, in which the modeler is interested in determining a solution that satisfies a constraint with some given probability. They cite two reasons for the continued intractability of these types of stochastic programming problems: (i) it is difficult to test whether a solution is feasible in the chance constraint, and (ii) the feasible region is not convex, implying that even if one could check for feasibility, checking for optimality remains difficult. Ahmed and Shapiro approximate the distribution that they use to model uncertainty of the probabilistic vector in their chance constraint with an empirical distribution based on a Monte Carlo sample. This approximated problem provides provably good solutions to the chance-constrained problem.

Wallace (2010) describes the way in which stochastic programming relates to the theory of real options. He sets forth the types of options, and states that the wait-and-see (scenario) solutions are unlikely to be those produced by a stochastic program. He concludes that a real options approach is not as general as a stochastic programming approach in that the former requires that the options must be identified a priori, and the latter produces the options as part of its solution.

Chen et al. (2008) present a tutorial in the area of simulation optimization. Unlike in stochastic programming, in which the underlying model is a mathematical program, the authors address the paradigm in which a simulation model is used to capture uncertain inputs with a view to optimizing performance of the system in question. The authors point out that the difficulties in this modeling paradigm stem from balancing the computational effort used to search for better solutions with the effort related to obtaining better estimates of the performance of the incumbent solution. The authors provide a context for and a description of existing solution approaches, and then treat three specific areas of this field: (i) allocating correctly simulation replications to determine an optimal solution, (ii) estimating an improving (but, necessarily, stochastic) search direction, and (iii) obtaining globally optimal solutions. They conclude with related work.

Bertsimas et al. (2011) discuss both methodology and computational results of robust optimization in which the goal is to determine an optimal solution for any realization of an outcome (where these outcomes are given as prespecified members of a set). The authors argue that many robust optimization problems are reasonably tractable, but that robust optimization may not be the appropriate modeling framework depending on the nature of uncertainty in the

problem. After developing mathematical models with various structures, and discussing how to model the uncertainty within these structures, some successful applications in finance, statistics, supply chain management, and engineering design are presented.

The stochastic programming community has a website (http://stoprog.org/) that provides resources such as tutorials (journal articles and presentations), papers, lecture notes, and books.

6. Specialized Solution Techniques

Although most of the discussion to this point has concerned algorithms and good modeling practice for monolithic formulations, the following authors offer polylithic solution techniques, i.e., techniques designed to partition a problem into manageable parts and solve these parts iteratively until convergence. In this section, we begin by describing exact techniques, specifically, two ways in which "difficult" (or many) variables can be handled, a method by which constraints are handled, and some specialized techniques; we conclude with heuristics.

In a seminal paper, Dantzig and Wolfe (1960) introduce a method that solves a large-scale linear program by examining only a subset of the columns (variables) of said linear program at a time; dual-price information is passed to subproblems and those subproblems generate columns that are passed back to a centralized master program for evaluation of their appeal in constituting the optimal solution. The authors begin with a specific linear programming format, show how the algorithm can handle generalized formats, and conclude by mentioning some specialized systems (e.g., block angular constraint sets) to which their approach can be applied. Subsequently, Lübbecke and Desrosiers (2005) survey contributions in Dantzig-Wolfe decomposition and column generation. They provide references for applications of column generation, and then outline the method. They describe the structure of integer programs and corresponding convexification principles. They then elaborate on column generation for integer programs. The authors explain the structure of the decomposed problem—the restricted master problem and the pricing problem. The article concludes with a discussion of implementation issues and their resolution, and practical guidelines for obtaining integer solutions from a procedure that solves linear programs. Feillet (2010) describes column generation as it applies specifically to the vehicle routing problem with time windows. The author provides a Dantzig-Wolfe reformulation to obtain a tighter linear programming relaxation, a branch-and-price strategy, cutting planes, and a Lagrangian procedure tailored to the specific application.

In another seminal paper, Benders (1962) describes a method for solving mixed-integer programming problems in which two classes of variables exist—those restricted to be integer, and those that are continuous. He places the former in an integer programming master problem and the latter in a linear programming subproblem. He shows two ways in which the subproblem can be solved—either via the primal or the dual formulation. He concludes with numerical results. Geoffrion (1972) generalizes this approach to the situation in which the subproblem is not a linear program, discusses cases in which this type of decomposition is appropriate, and gives preliminary computational results.

The above techniques address the way in which a problem can be made more tractable by reducing the number of variables that must simultaneously be considered, either in that said variables are introduced into a problem in stages, or that they are partitioned into separate problems and their values determined in their respective subproblem. Lagrangian relaxation also partitions the problem, but by relaxing constraints into the objective function with a penalty for their violation. The eminently readable and highly cited paper by Fisher (1981) describes this technique, and provides a numerical example of how lower and upper bounds on the problem instance are derived. The example consists of a generalized assignment model in which the knapsack constraints are relaxed into the objective. The author also discusses multiplier updates and the shortcomings of the technique.

Kallrath (2011) provides summaries of polylithic solution approaches, i.e., solution approaches that decompose the original problem and solve it iteratively, in parts. The author also provides guidance on careful model formulation and constructs, e.g., SOS-2 constraints, that may be helpful in solving mathematical programs either in their monolithic or in their polymathic format.

In the realm of heuristic techniques, Glover (1990) provides a fundamental tutorial on tabu search, which he defines as the transformation of one solution into another for the purpose of evaluating the solution with respect to an objective (or measure of "goodness"). A transformation consists of a sequence of moves, which could assume the form of the addition or removal of a variable in the solution, or a change in the solution's variable value(s). He provides examples of successful applications of tabu search, and emphasizes and explains certain characteristics of the search, such as its short-term memory mechanisms of identifying good candidate solutions. Glover (1990) carefully illustrates the search procedure with an example of a modified minimum cost spanning tree. The author then discusses longer term memory implications as they relate to search intensification strategies. Schneider (2011) describes a specific

application of tabu search regarding the manufacture of cables for cars; there are ramping and retooling times associated with setups, which are reflected in the overall cost of producing cables. Certain jobs can only be assigned to certain machines, where a job consists of a type of cable, differentiated by its length, diameter, color, and insulation material. The author provides a basic tabu search algorithm, and then shows how this algorithm can be tailored for her problem, specifically, with a view to enhancing computational efficiency. The problem the author presents is general enough that researchers can benefit from the ideas for a different problem setting.

7. Modeling Languages and Solvers

Bixby (2012) gives an amusing, yet comprehensive and easily understandable, history of computation for both linear and integer programming problems, from well before the advent of modeling languages, efficient computer codes, and hardware and programming as we know them today. A practitioner can now solve an optimization model through direct use of commercial or open-source software containing the linear and integer programming algorithms described above (or with the use of a customized algorithm). However, the software requires an interface that accepts vectors and matrices of data, specifically, vectors containing objective function coefficients and constraint data in the form of matrices of left-hand-side coefficients and vectors of right-hand-side constants. Implementation of the specialized solution techniques described in §6 requires iterative solutions of subproblems, and information passing between the iterations. Programming languages such as C++ can be used to generate vectors and matrices in the form required by optimization solvers, and allow for "scripting" or writing procedural code to solve parts of a monolith and pass information from that solve to subsequent ones. However, decades ago, researchers recognized that the time required to write the necessary code and the inflexibility of the resulting code would lead many practitioners to adopt modeling languages, or higher-level, specialized programming constructs suited for optimization model formulation, e.g., set constructs, and the expression of objective functions and constraints similar to the way in which they appear when written mathematically. These modeling languages allow for more rapid model development, and for more flexibility in changing and debugging a model than conventional programming languages do. Fourer (2012) provides a history of matrix generators, motivating their need, discussing their correspondence with modeling languages, and providing examples of syntax. Among those algebraic modeling languages Fourer (2012) mentions, the most commonly used ones for linear and mixed-integer programming problems are AMPL (AMPL 2009, Fourer et al. 2003) and GAMS (GAMS 2012, Rosenthal 2012). The most competitive linear and mixed-integer programming solvers at the time of this writing are CPLEX (IBM 2009), GuRoBi (GuRoBi 2009), and Xpress (FICO 2008). CPLEX, GuRoBi, and Xpress can all accept models written in either AMPL or GAMS; the Xpress solver can also accept models written in its Mosel modeling language (FICO 2008). (CPLEX also has its own modeling language, OPL.) Unlike AMPL and GAMS, Mosel is a compiled language, making it faster to read into the solver for large models, which AMPL and GAMS only interpret. However, Mosel has the disadvantage that it lacks solver versatility, i.e., at the time of this writing, the solvers with which the modeling language is compatible are more limited. AMPL and GAMS, on the other hand, can be used with many solvers (including a variety of nonlinear ones). Kallrath (2012) describes algebraic modeling languages including, but not limited to, the ones we mention.

It is now possible to obtain heavily discounted or even free academic licenses for most modeling languages and solvers, which possess the full capability of commercial software. However, these licenses can only be used for academic (i.e., teaching and research) purposes. It is also possible to access solvers on the NEOS server, see http://www.neos-server.org/neos/, hosted by the University of Wisconsin, Madison. Most linear and mixed-integer solvers on this website accept AMPL and/or GAMS input. Solvers (and other software including a modeling language called FlopC++) are also available on the COIN-OR website (http://www.coin-or.org/index.html) as part of the COmputational INfrastructure for Operations Research project designed to encourage the development and improvement of open-source software (see Martin 2010). Sandia National Laboratories has been at the forefront of developing a Python-based open-source optimization package with modeling language capabilities, and solvers that take advantage of special problem structure both in stochastic and deterministic (integer) programs (https://software.sandia.gov/trac/coopr/).

8. Applications

In addition to focusing on theory and algorithms, many researchers have written tutorials on well known applications, in some cases, furthering the theory in addressing the application. For example, Bertsimas and Stock Patterson (1998) describe the "air traffic flow management problem," involving routing aircraft through capacity-constrained sectors in the sky. The authors use a clever formulation for

their integer programming problem with an underlying network structure and show how the resulting structure possesses facet-defining constraints, greatly enhancing model tractability. This research provides an excellent example of how unintuitive model reformulation can lead to dramatic improvements in computational performance. Lambert et al. (2013) describe a prominent class of problems in open pit mining, the open pit production scheduling problem and its variants. Lambert et al. (2013) focus on computational aspects of certain applied models. The authors discuss variable elimination techniques, variable definitions, strong formulations, and techniques for quickly obtaining integer feasible solutions. In many cases, these ideas existed in the literature, but had never been carefully written down and explained mathematically. The observant reader should be able to apply the insights in the Lambert article to models with a similar precedence-constrained knapsack structure.

Arguably, operations research began as a field with its applications during World War II. Sheldon and Yost (2011) lead the reader through four decades of a mathematical modeler, algorithmic expert, and practitioner as he recounts his (at the time of this writing, still active) life in military operations research. Not only does this article provide insights into military modeling and the way in which the U.S. military uses operations research, but it also points out the advances in and shortcomings of the field. For example, the article mentions the important modeling concept of elastic programming, and also the mysterious concept of the "subject matter expert," noting that there are no "subject matter apprentices," and issuing the caveat that the former designation seems almost always to be a subjective one.

An excellent tutorial on an application with both military and civilian relevance is given by Brown et al. (2006), who introduce bi-and tri-level models whose solutions dictate how to fortify assets in the face of attack from an "intelligent" terrorist (or, in a civilian setting, how to respond to natural disasters). The authors begin by listing U.S.-defined critical assets, and note that these are often characterized by the attributes of criticality, vulnerability, reconstitutability, and threat. They then introduce an attacker-defender optimization model in which an inner problem specifies that a "defender" wishes to utilize his asset most efficiently, whereas an "attacker" seeks to heavily damage or destroy the asset under the assumptions that the attacker has perfect information regarding the way in which the defender uses his (damaged) asset and is capable of optimally manipulating the system in his favor. The paper also details related model types, i.e., defender-attacker models and defender-attacker-defender models, discusses the mathematics and solution techniques behind each, and concludes with examples.

In recent applications that are somewhat unconventional for operations researchers, but interesting nonetheless, Worden et al. (2011) and Cohen and Parhi (2011) provide tutorials on soft algorithms for mechanical systems, and on optimization of the RSA public key cryptosystem, respectively.

Finally, Brown and Rosenthal (2008) discuss "secrets" for success in applied optimization modeling, which include guidance on: (i) writing and model formulation, (ii) the realization of a fine line between what classifies as an objective function and what classifies as a constraint, (iii) the necessity of running a variety of model instances, (iv) the benefits of modeling robustly, (v) the insights available from the dual model, (vi) the recognition of the benefits and drawbacks of modeling languages and spreadsheets, (vii) the strengths and weaknesses of sensitivity analysis and heuristics, and (viii) the importance of clearly conveying model results.

9. Optimization Textbooks

Although we provide here a tutorial consisting of journal articles, technical reports, Web pages, and manuals containing advanced material not necessarily found in books, we mention in closing books that may prove to be helpful references: (i) Winston and Goldberg (2004) provide an advanced undergraduate and beginning masters-level treatment of many operations research topics; Rardin (1998) addresses the same for optimization modeling in particular; and Kallrath and Wilson (1997) address linear and mixed-integer optimization models and solution algorithms specifically; (ii) Dantzig (1963) represents a seminal linear programming text, with updates given in Dantzig and Thapa (1997, 2003); (iii) Bertsimas and Tsitsiklis (1997) address both linear and integer optimization, as do Wolsey (1998), Martin (1999) and Bertsimas and Weismantel (2005); (iv) Bazaraa et al. (2005) and Ahuja et al. (1993) provide coverage of network flow models, with the former reference also including linear programming topics. Finally, in the stochastic programming domain, King and Wallace (2012) emphasize modeling, while Shapiro et al. (2009), Birge and Louveaux (1997), and Prekopa (1995) focus on theory and algorithms. Although this section in no way provides a complete listing of useful books on the topics of linear and mixed-integer optimization, we attempt to highlight texts that advanced undergraduate and graduate students may find useful.

10. Summary and Conclusions

We have surveyed seminal works and tutorials on linear and mixed-integer programming. Tables 1 and 2

Table 1 Summary of Cited Work on Linear Programming, Mixed-Integer Programming, and Optimization Under Uncertainty

Article	Topic	Subtopic	Article type	Article level†
Dantzig et al. (1955)	Linear programming	Primal simplex	Seminal paper	I
Lemke (1954)	Linear programming	Dual simplex	Seminal paper	A
Klee and Minty (1972)	Linear programming	Complexity	Seminal paper	A
Khachian (1979)	Linear programming	Ellipsoid method	Seminar paper	A
Karmarkar (1984)	Linear programming	Interior point	Seminal paper	A
Gill et al. (1986)	Linear programming	Method and performance contrast	Seminal paper	A
Marsten et al. (1990)	Linear programming	Interior point motivation	Tutorial	I
Lustig et al. (1994)	Linear programming	Interior point implementation	State-of-the-art	A
Terlaky (2009)	Linear programming	Interior point review	State-of-the-art	A
Murty (2009)	Linear programming	Sphere methods	Tutorial	A
Klotz and Newman (2013a)	Linear programming	Implementation	Tutorial	I
Cook (2012)	Integer programming	Background and history	Survey	B
Hoffman (2000)	Integer programming	Large-scale improvements	State-of-the-art	I
Bixby et al. (2000)	Integer programming	Progress	State-of-the-art	I
Bixby and Rothberg (2007)	Integer programming	Progress	State-of-the-art	I
Camm et al. (1990)	Integer programming	Big "*M*"	Tutorial	B
Trick (1997)	Integer programming	Examples	Tutorial	B
Brown and Dell (2007)	Integer programming	Formulation	Tutorial	B
Trick (2005)	Integer programming	Formulations	Tutorial	I
Savelsbergh (1994)	Integer programming	Preprocessing	Tutorial	A
Sherali and Smith (2001)	Integer programming	Symmetry	Tutorial	A
Brown et al. (1997)	Integer programming	Persistence	Tutorial	I
Rebennack et al. (2012)	Integer programming	Maximum stable set	Tutorial	A
Cornuéjols (2008)	Integer programming	Cuts	Tutorial	A
Cornuéjols (2012)	Integer programming	Gomory cuts	History	B
Danna (2008)	Integer programming	Performance variability	Tutorial	A
Klotz and Newman (2013b)	Integer programming	Implementation	Tutorial	I
Hooker (1994)	Integer programming	Constraint programming	Tutorial	A
Smith (1995)	Integer programming	Constraint satisfaction	Tutorial	A
Brailsford et al. (1999)	Integer programming	Constraint satisfaction	Tutorial	A
Lustig and Puget (2001)	Integer programming	Constraint programming	Tutorial	I
Dantzig (1955)	Stochastic programming	Math structure	Seminal paper	A
Beale (1955)	Stochastic programming	Math structure	Seminal paper	A
Sen and Higle (1999)	Stochastic programming	Recourse	Tutorial	I
Higle (2005)	Stochastic programming	Comparisons with determinism	Tutorial	A
Rockafellar (2007)	Optimization under uncertainty	Risk	Tutorial	A
Ahmed and Shapiro (2008)	Stochastic programming	Chance constraints	Tutorial	A
Wallace (2010)	Optimization under uncertainty	Real options	Tutorial	A
Chen et al. (2008)	Optimization under uncertainty	Simulation	Tutorial	A
Bertsimas et al. (2011)	Robust optimization	Methodology and modeling	Tutorial	A

†Article levels: beginning (B), intermediate (I), and advanced (A).

summarize the sources we have cited. It is our hope that advanced undergraduate and graduate students can use our paper and the references contained herein in order to fill the gaps between course work and their own research.

Acknowledgments

The authors wish to thank Holly Graham (Colorado School of Mines) for her help with retrieving reference material and with bibliographic formatting. The views expressed here are those of the authors and may not be considered as an official position of the European Commission.

References

Ahmed S, Shapiro A (2008) Solving chance-constrained stochastic programs via sampling and integer programming. Chen Z, Raghavan S, eds. *Tutorials in Operations Research* (INFORMS, Hanover, MD), 261–269.

Ahuja R, Magnanti T, Orlin J (1993) *Network Flows: Theory, Algorithms, and Applications* (Prentice Hall, Englewood Cliffs, NJ).

AMPL (2009) *AMPL—A Mathematical Programming Language* (AMPL Optimization LLC, Albuquerque, NM).

Bazaraa M, Jarvis J, Sherali H (2005) *Linear Programming and Network Flows* (John Wiley & Sons, Inc., New York).

Bazaraa M, Sherali H, Shetty C (2006) *Nonlinear Programming: Theory and Algorithms* (John Wiley & Sons, Inc., Hoboken, NJ).

Beale EML (1955) On minimizing a convex function subject to linear inequalities. *J. Roy. Statist. Soc.* 17(2):173–184.

Benders J (1962) Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4(3):238–252.

Bertsimas D, Stock Patterson S (1998) The air traffic flow management problem with enroute capacities. *Oper. Res.* 46(3):406–422.

Bertsimas D, Tsitsiklis J (1997) *Introduction to Linear Optimization* (Athena Scientific, Belmont, MA).

Bertsimas D, Weismantel R (2005) *Optimization Over Integers* (Dynamic Ideas, Belmont, MA).

Table 2 Summary of Cited Work on Decomposition Techniques, Modeling Languages and Solvers, and Applications

Article	Topic	Subtopic	Article type	Article level†
Dantzig and Wolfe (1960)	Tailored algorithms	Dantzig-Wolfe decomposition	Seminal paper	A
Lübbecke and Desrosiers (2005)	Tailored algorithms	Column generation	Tutorial	A
Feillet (2010)	Tailored algorithms	Column generation (for vrp*)	Tutorial	A
Benders (1962)	Tailored algorithms	Benders decomposition	Seminal paper	A
Geoffrion (1972)	Tailored algorithms	Generalized Benders decomposition	Seminal paper	A
Fisher (1981)	Tailored algorithms	Lagrangian relaxation	Seminal paper	I
Kallrath (2011)	Tailored algorithms	Polylithic approaches	Tutorial	I
Glover (1990)	Tailored algorithms	Tabu search	Tutorial	I
Schneider (2011)	Tailored algorithms	Tabu search (example)	Tutorial	I
Bixby (2012)	Computation	History	Tutorial	B
Fourer (2012)	Software	History	Survey	B
Fourer et al. (2003)	Software	AMPL	Manual	B
Rosenthal (2012)	Software	GAMS	Manual	B
Kallrath (2012)	Software	Modeling language description	Tutorial	B
Martin (2010)	Software	COIN-OR	Manual	B
Bertsimas and Stock Patterson (1998)	Application	Air traffic flow management	Seminal paper	I
Lambert et al. (2013)	Application	Mining	Tutorial	I
Sheldon and Yost (2011)	Application	Military	Interview	B
Brown et al. (2006)	Application	Infrastructure defense	Tutorial	I
Worden et al. (2011)	Application	Algorithms for mechanical systems	Tutorial	I
Cohen and Parhi (2011)	Application	RSA public key cryptosystem	Tutorial	I
Brown and Rosenthal (2008)	Applications	Secrets for success	Tutorial	B

†Article levels: beginning (B), intermediate (I), and advanced (A).
*Vehicle routing problem.

Bertsimas D, Brown D, Caramanis C (2011) Theory and applications of robust optimization. *SIAM Rev.* 53(3):464–501.
Birge J, Louveaux F (1997) *Introduction to Stochastic Programming* (Springer, New York).
Bixby R (2012) A brief history of linear and mixed-integer programming. Grötschel M, ed. *Documenta Mathematica—Optim. Stories, 21st Internat. Sympos. Math. Programming* (Journal der Deutschen Mathematiker-Vereinigung, Berlin), 107–121.
Bixby R, Rothberg E (2007) Progress in computational mixed integer programming—A look back from the other side of the tipping point. *Ann. Oper. Res.* 149(1):37–41.
Bixby R, Fenelon M, Gu Z, Rothberg E, Wunderling R (2000) MIP: Theory and Practice: Closing the gap. *System Modeling and Optimization: Methods, Theory and Applications* (Kluwer, The Netherlands), 19–50.
Brailsford S, Potts C, Smith B (1999) Constraint satisfaction problems: Algorithms and applications. *Eur. J. Oper. Res.* 119(3):557–581.
Brown G, Carlyle M, Salmerón J, Wood K (2006) Defending critical infrastructure. *Interfaces* 36(6):530–544.
Brown G, Rosenthal R (2008) Optimization tradecraft: Hard-won insights from real-world decision support. *Interfaces* 38(5): 356–366.
Brown G, Dell R (2007) Formulating integer linear programs: A rogues' gallery. *INFORMS Trans. Ed.* 7(2):1–13.
Brown G, Dell R, Wood RK (1997) Optimization and persistence. *Interfaces* 27(5):15–37.
Camm J, Raturi A, Tsubakitani S (1990) Cutting big *M* down to size. *Interfaces* 20(5):61–66.
Chen C-H, Fu M, Shi L (2008) Simulation and optimization. Chen Z, Raghavan S, eds. *Tutorials in Operations Research* (INFORMS, Hanover, MD), 247–260.
Cohen A, Parhi K (2011) Architecture optimizations for the RSA public key cryptosystem: A tutorial. *IEEE Circuits and Systems Magazine* 11(4):24–34.
Cook W (2012) Markowitz and Manne + Eastman + Land and Doig = branch and bound. Grötschel M, ed. *Documenta Mathematica—Optim. Stories, 21st Internat. Sympos. Math. Programming*, (Journal der Deutschen Mathematiker-Vereinigung, Berlin), 227–238.
Cornuéjols G (2008) Valid inequalities for mixed integer linear programs. *Math. Programming* 112(1):3–44.
Cornuéjols G (2012) The ongoing story of Gomory cuts. Grötschel M, ed. *Documenta Mathematica—Optim. Stories. 21st Internat. Sympos. Math. Programming* (Journal der Deutschen Mathematiker-Vereinigung, Berlin), 221–226.
Danna E (2008) Performance variability in mixed integer programming. MIP 2008 Workshop, Columbia University, New York, http://www.iro.umontreal.ca/~gendron/IFT6551/LECTURES/Computation.pdf.
Dantzig G (1955) Linear programming under uncertainty. *Management Sci.* 1(3–4):197–206.
Dantzig G (1963) *Linear Programming and Extensions* (Princeton University Press, Princeton, NJ).
Dantzig G, Thapa M (1997) *Linear Programming* 1: *Introduction* (Springer, New York).
Dantzig G, Thapa M (2003) *Linear Programming* 2: *Theory and Extensions* (Springer, New York).
Dantzig G, Wolfe P (1960) Decomposition principle for linear programs. *Oper. Res.* 8(1):101–111.
Dantzig G, Orden A, Wolfe P (1955) A generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific J. Math.* 5(2):183–195.
Feillet D (2010) A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR: Quart. J. Oper. Res.* 8(4):407–424.
FICO (2008) Xpress-MP Optimization Suite. http://www.fico.com/en/Products/OMTools/Pages/FICO-Xpress-Optimization-Suite.aspx, Minneapolis.
Fisher M (1981) The Lagrangian relaxation method for solving integer programming problems. *Management Sci.* 27(1):1–18.
Fourer R (2012) On the evolution of optimization modeling systems. Grötschel M, ed. *Documenta Mathematica—Optim. Stories, 21st Internat. Sympos. Math. Programming* (Journal der Deutschen Mathematiker-Vereinigung, Berlin), 377–388.

Fourer R, Gay D, Kernighan B (2003) *AMPL—A Modelling Language for Mathematical Programming* (Thomson Brooks/Cole, Pacific Grove, CA).

GAMS (2012) GAMS Distribution 23.9.1. Washington, DC.

Gass S, Assad A (2011) History of operations research. Geunes J, ed. *Tutorials in Operations Research* (INFORMS, Hanover, MD), 1–14.

Geoffrion A (1972) Generalized Benders decomposition. *J. Optim. Theory Appl.* 10(4):237–260.

Gill P, Murray W, Saunders M, Tomlin J, Wright M (1986) On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Math. Programming* 36(2):183–209.

Glover F (1990) Tabu research: A tutorial. *Interfaces* 20(4):74–94.

Grötschel M (2012) *Documenta Mathematica—Optim. Stories, 21st Internat. Sympos. Math. Programming* (Journal der Deutschen Mathematiker-Vereinigung, Berlin).

GuRoBi (2009) *GuRoBi Optimizer* (GuRoBi Optimization Inc., Houston).

Higle J (2005) Stochastic programming: Optimization when uncertainty matters. Smith JC, ed. *Tutorials in Operations Research* (INFORMS, Hanover, MD), 30–53.

Hoffman K (2000) Combinatorial optimization: History and future challenge. *J. Appl. Comput. Math.* 124(1–2):341–360.

Hooker J (1994) Logic-based methods for optimization. Borning A, ed. *Principles and Practice of Constraint Programming*, Vol. 874, Lecture Notes in Computer Science (Springer, Berlin, Heidelberg), 336–349.

IBM (2009) ILOG CPLEX IBM—International Business Machines Corporation, Incline Village, NV.

Johnson D (2012) A brief history of NP-completeness, 1954–2012. Grötschel M, ed. *Documenta Mathematica—Optim. Stories. 21st Internat. Sympos. Math. Programming* (Journal der Deutschen Mathematiker-Vereinigung, Berlin), 359–376.

Kallrath J (2011) Polylithic modeling and solution approaches using algebraic modeling systems. *Optim. Lett.* 5(3):453–466.

Kallrath J (2012) *Algebraic Modeling Systems—Modeling and Solving Real World Optimization Problems* (Springer-Verlag, Heidelberg, Germany).

Kallrath J, Wilson JM (1997) *Business Optimisation Using Mathematical Programming* (Macmillan Press, London).

Karmarkar N (1984) A new polynomial-time algorithm for linear programming. *Combinatorica* 4(4):373–395.

Khachian L (1979) A polynomial algorithm in linear programming. *Soviet Math. Doklady* 20:191–194.

King A, Wallace S (2012) *Modeling with Stochastic Programming* (Springer Series in Operations Research and Financial Engineering, New York).

Klee V, Minty G (1972) How good is the simplex algorithm? Shisha O, ed. *Inequalities III* (Academic Press, New York), 159–175.

Klotz E, Newman A (2013a) Practical guidelines for solving difficult linear programs. *Surveys Oper. Res. Management Sci.* 18(1–2): 1–17.

Klotz E, Newman A (2013b) Practical guidelines for solving difficult mixed integer programs. *Surveys Oper. Res. Management Sci.* 18(1–2):18–32.

Lambert W, Brickey A, Eurek K, Newman A (2013) Open pit block sequencing formulations: A tutorial. *Interfaces*. Forthcoming.

Lee J, Leyffer S (2012) *Mixed Integer Nonlinear Programming, the IMA Volumes in Mathematics and Its Applications*, Vol. 154 (Springer, New York).

Lemke C (1954) The dual method of solving the linear programming problem. *Naval Res. Logist. Quart.* 1(1):36–47.

Lübbecke M, Desrosiers J (2005) Selected topics in column generation. *Oper. Res.* 53(6):1007–1023.

Lustig I, Puget J-F (2001) Program does not equal program: Constraint programming and its relationship to mathematical programming. *Interfaces* 31(6):29–53.

Lustig I, Marsten R, Shanno D (1994) Interior-point methods for linear programming: Computational state of the art. *ORSA J. Comput.* 6(1):1–14.

Marsten R, Subramanian R, Saltzman M, Lustig I, Shanno D (1990) Interior point methods for linear programming: Just call Newton, Lagrange, and Fiacco and McCormick! *Interfaces* 20(4):105–116.

Martin K (1999) *Large Scale Linear and Integer Optimization: A Unified Approach* (Kluwer Academic Publishers, Boston).

Martin K (2010) Tutorial: COIN-OR: Software for the OR community. *Interfaces* 40(6):465–476.

Murty K (2009) New sphere methods for linear programs. Oskoorouchi M, ed. *Tutorials in Operations Research* (INFORMS, Hanover, MD), 62–80.

Němeček J, Němečková H (2012) The origins of minimal spanning tree algorithms—Borůvka and Jarník. Grötschel M, ed. *Documenta Mathematica—Optim. Stories. 21st Internat. Sympos. Math. Programming* (Journal der Deutschen Mathematiker-Vereinigung, Berlin), 127–141.

Powell W (2009) What you should know about approximate dynamic programming. *Naval Res. Logist.* 56(3):239–249.

Prekopa A (1995) *Stochastic Programming* (Kluwer Academic Publishers, Norwell, MA).

Rardin R (1998) *Optimization in Operations Research* (Prentice Hall, Upper Saddle River, NJ).

Rebennack S, Reinelt G, Pardalos P (2012) A tutorial on branch and cut algorithms for the maximum stable set problem. *Internat. Trans. Oper. Res.* 19(1–2):161–199.

Rockafellar R (2007) Coherent approaches to risk in optimization under uncertainty. Klastorin T, ed. *Tutorials in Operations Research* (INFORMS, Hanover, MD), 38–61.

Rosenthal R (2012) GAMS: A User's Guide, http://www.gams.com/dd/docs/bigdocs/GAMSUsersGuide.pdf, Washington, DC.

Savelsbergh M (1994) Preprocessing and probing techniques for mixed integer programming problems. *ORSA J. Comput.* 6(4):445–454.

Schneider U (2011) A tabu search tutorial based on a real-world scheduling problem. *Central Eur. J. Oper. Res.* 19(4):467–493.

Schrijver A (2012a) On the history of the shortest path problem. Grötschel M, ed. *Documenta Mathematica—Optim. Stories, 21st Internat. Sympos. Math. Programming* (Journal der Deutschen Mathematiker-Vereinigung, Berlin), 155–167.

Schrijver A (2012b) On the history of the transportation and maximum flow problems. Grötschel M, ed. *Documenta Mathematica—Optim. Stories, 21st Internat. Sympos. Math. Programming* (Journal der Deutschen Mathematiker-Vereinigung, Berlin), 169–180.

Sen S, Higle J (1999) An introductory tutorial on stochastic linear programming models. *Interfaces* 29(2):33–61.

Shapiro A, Dentcheva D, Ruszczynski A (2009) *Lectures on Stochastic Programming: Modeling and Theory* (Society for Industrial and Applied Mathematics and the Mathematical Programming Society, Philadelphia).

Sheldon B, Yost K (2011) Military operations research society (MORS): Oral history project interview of Dr. Gerald G. Brown. *Military Oper. Res.* 16(4):57–82.

Sherali D, Smith J (2001) Improving discrete model representations via symmetry considerations. *Management Sci.* 47(10): 1396–1407.

Smith BM (1995) A tutorial on constraint programming. Report 95.14. University of Leeds. http://www.dcs.gla.ac.uk/~pat/cp4/papers/95_14.pdf.

Terlaky T (2009) Twenty-five years of interior point methods. Oskoorouchi M, ed. *Tutorials in Operations Research* (INFORMS, Hanover, MD), 1–33.

Tovey C (2002) Tutorial on computational complexity. *Interfaces* 32(3):30–61.

Trick M (1997) A tutorial on integer programming. Carnegie Melon University, http://mat.gsia.cmu.edu/orclass/integer/integer.html.

Trick M (2005) Formulations and reformulations in integer programming. Carnegie Melon University, http://mat.gsia.cmu.edu/trick/formul04.pdf.

Wallace S (2010) Stochastic programming and the option of doing it differently. *Ann. Oper. Res.* 177(1):3–8.

Winston W, Goldberg J (2004) *Operations Research: Applications and Algorithms* (Duxbury Press, Belmont, CA).

Wolsey L (1998) *Integer Programming* (John Wiley & Sons, New York).

Worden K, Staszewski W, Hensman J (2011) Natural computing for mechanical systems research: A tutorial overview. *Mech. Systems and Signal Processing* 25(1):4–111.