Chapter

Recent Advancements in Commercial Integer Optimization Solvers for Business Intelligence Applications

Cheng Seong Khor

Abstract

The chapter focuses on the recent advancements in commercial integer optimization solvers as exemplified by the CPLEX software package particularly but not limited to mixed-integer linear programming (MILP) models applied to business intelligence applications. We provide background on the main underlying algorithmic method of branch-and-cut, which is based on the established optimization solution methods of branch-and-bound and cutting planes. The chapter also covers heuristic-based algorithms, which include preprocessing and probing strategies as well as the more advanced methods of local or neighborhood search for polishing solutions toward enhanced use in practical settings. Emphasis is given to both theory and implementation of the methods available. Other considerations are offered on parallelization, solution pools, and tuning tools, culminating with some concluding remarks on computational performance vis-à-vis business intelligence applications with a view toward perspective for future work in this area.

Keywords: integer programming, valid inequalities, local branching, relaxation induced neighborhood search (RINS), evolutionary algorithms, solution polishing

1. Introduction

The ongoing drive on Industrial Revolution 4.0 particularly to take advantage of big data analytics has impacted business intelligence applications significantly spanning various areas including resource assessment, corporate development, and advanced technology R&D research and development [1]. A key enabler supporting the transformation to digitalization is optimization technology which encompasses the established methodologies of linear and nonlinear programming with extensions to discrete or integer programming. This chapter focuses on recent advancements in commercial optimization solvers notably the industry-leading software package of IBM ILOG CPLEX [2] as applied to variants of integer programming problems particularly mixed-integer linear programming (MILP) models.

This chapter aims to contribute towards highlighting the growing and maturing capability of integer optimization especially in the last decade or so towards addressing, solving, analyzing, and eliciting insights from practical business intelligence applications. With rapid developments in the realm of big data analytics as spurred by Industry Revolution 4.0, advancement in optimization technology including integer optimization is imperative to support if not spearhead the changes at the forefront of the transformation taking place. The rest of the chapter is organized as follows. Section 2 gives an overview of the present role of integer optimization in business intelligence applications. Major solution methods and algorithms with certain enhanced features typically available in standard integer optimization solvers are detailed in Section 3 including those intended to exploit model formulations. Section 4 describes and discusses several real-world use cases on practical business intelligence applications that illustrate the applicability and strengths of integer optimization solvers. Finally, concluding remarks on the salient features of standard integer optimization solvers for business intelligence applications are offered including perspectives for future research directions.

2. Overview of integer optimization in business intelligence applications

Numerous business intelligence applications can be posed as mathematical programming problems that can be handled by commercial optimization solvers such as CPLEX, Gurobi [3], or KNITRO [4]. The problems can be formulated as models that include linear programming (LP), mixed-integer linear programming (MILP), quadratic programming (QP), mixed-integer quadratic programming (MIQP), quadratically-constrained programming, and mixed-integer quadratically-constrained programming. Such solvers are also used in tandem with other appropriate optimization solvers to handle other mainly nonlinear problems such as mixed-integer nonlinear programming (MINLP) models or in general, mixed-integer programs (MIP) [5].

2.1 Computational performance of commercial integer optimization solvers

The actual computational performance of a commercial optimizer (or optimization package) such as CPLEX results from a combination of improvement in several aspects. They include LP solvers with capability and features including preprocessing, algebra for sparse systems, solution methods (primal or dual simplex and barrier), and techniques to overcome degeneracy and numerical difficulties [6]. Equally important is the use of cutting planes as valid inequalities in solving problems that bridges the gap from theory to practice [7]. Further improvement involves applying heuristics including node heuristics (e.g., local branching, guided dives) and relaxation-induced neighborhood search, invoking evolutionary algorithms for solution polishing; and implementing parallelization for efficient computations [8].

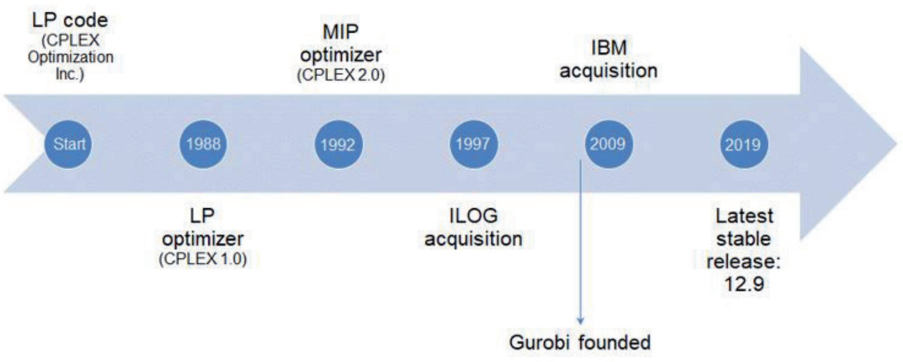

Figure 1.
Historical background of IBM ILOG CPLEX integer optimization solver.

Year	Activity/accomplishment
1988	Develops LP solver (CPLEX 1.0)
1992	Offers simple branch and bound with limited cuts (CPLEX 2.0)
1998	Incorporates simple heuristic; provides faster dual simplex (CPLEX 6.0)
1999	Introduces five node heuristics and six cutting plane types (CPLEX 6.5)
2000	Caters for semi-continuous and semi-integer variables; stipulates dual simplex as default LP solution method; introduces preprocessing; improved cuts (CPLEX 7.0)
2002	Introduces new LP method of sifting, concurrent optimization, new QP capabilities, and 9 cutting plane types (CPLEX 8.0)
2003	Introduces quadratic constraint programming (QCP) and relaxation induced neighborhood search (RINS) (CPLEX 9.0)
2006	Improved MIQP, changes in MIP start, feasible relaxation; introduces indicators and solution polishing features
2007	CPLEX 11.0 incorporates solution pool, tuning pool, and parallel mode
2010	Offers faster MILP solution; introduces multicommodity flow cuts; enhanced heuristics and dynamic search (in CPLEX 12.2)
2017	Enables faster MILP solution; enhanced CP Optimizer; new callback framework (in CPLEX 12.8)
2019	Includes handling of multiobjective problems; provides modeling assistance (in CPLEX 12.9)

Table 1.
Software release history of IBM ILOG CPLEX integer optimization solver.

2.2 A commercial success story: CPLEX integer optimization solver

CPLEX is a state-of-the-art commercial integer optimization solver currently marketed by IBM. It represents an early commercial success story of an optimization package with various acquisitions and a spin-off solver (called Gurobi) which is now a success story of its own. **Figure 1** presents brief historical facts of CPLEX while **Table 1** summarizes the software release history.

3. Solution methods and algorithms

3.1 Integer optimization algorithms

A suite of algorithms is available in various integer optimization solver to exploit the underlying problem structure of a business intelligence application towards achieving efficiency and accuracy. **Table 2** summarizes the typical main algorithms employed by CPLEX according to the problem type identified together with remarks on the enhancement provided to increase computational performance [9].

3.2 Branch-and-bound

A general structure of mixed-integer program is given by:

minimize

$$c^T x \tag{1}$$

subject to

$$Ax = b \tag{2}$$

$$l \le x \le u \tag{3}$$

$$\text{some } x \text{ are integers.} \tag{4}$$

Branch-and-bound is a base algorithm to solve MIP which uses LP as a subroutine [10]. The key strategies of a branch and bound procedure involve splitting (i.e., branching) the solution space into disjoint subspaces, bounding the objective function values for all solutions in the subspaces, and pruning or fathoming nodes of branches that cannot yield better solutions. Although it is provably exponential in time, tricks are available to accelerate its search which mostly apply to a subset of models with a suite of algorithms available.

The branching strategies are performed on the integer variables and comprise two main steps: (1) Choose an integer variable as a branching variable x_j, (2) Split the problem into two submodels: $x_j \le i$ or $x_j \ge i+1$ where for the special case of binary variables, the problem becomes $x_j = 0$ or $x_j = 1$.

The bounding problem given by the continuous (LP) relaxation to determine a lower bound z_{IP}^{L} on the objective function value of the original MIP problem can be described as follows: minimize $c^T x \left(= z_{IP}^{L}\right)$ subject to $Ax = b$, $l \le x \le u$ (simple bounds), and some x_j are integers. The continuous relaxation problem gives solution of an optimal objective value of z_{IP}^{L}, which is a lower bound on the objective function value of the original MIP problem by relaxing the integrality restriction. There are two useful properties of continuous relaxation: (1) If its solution satisfies integrality restrictions, there is no need to further explore the subspace; (2) It offers natural branching candidates as the integer variables with fractional values in a relaxation solution.

Key steps in the branch-and-bound procedure are summarized in **Figure 2**. As described in **Figure 2**, node selection in step 1 involves a tradeoff between achieving feasibility and optimality. The options available for node selection include depth first, breadth first, best first, limited discrepancy, and best estimate. When exploring nodes deep in a search tree, one is more likely to find integer feasible solutions and explore nodes that would be pruned by later feasible solutions. The method called plunging (as combined with those aforementioned) always choose a child node of previously explored node.

In step 2, the node relaxation step is ideally suited to dual simplex method. It involves only a small change from the parent relaxation solution (at the root node) and gives a new bound on the branching variable while maintaining dual feasibility of the previous basis. Thus, the solution is likely to be close to the previous basis.

Solver/optimizer	Algorithm	Model type	Remark
Simplex	Primal, dual, network	LP, QP	Reoptimization with simplex algorithms is faster when starting from a previous basis
Barrier	Interior-point	LP, QP, QCP	Explore multiple threads presence Barrier optimizer cannot start from advanced basis—limited application in B&B for MILP
Mixed-integer optimizers	Branch and cut, dynamic search	MILP, MIQP, MIQCP	IBM proprietary/trade secret methodology to solve MIP (some user callbacks cannot be used)

Table 2.
Algorithms available in IBM ILOG CPLEX integer optimization solver.

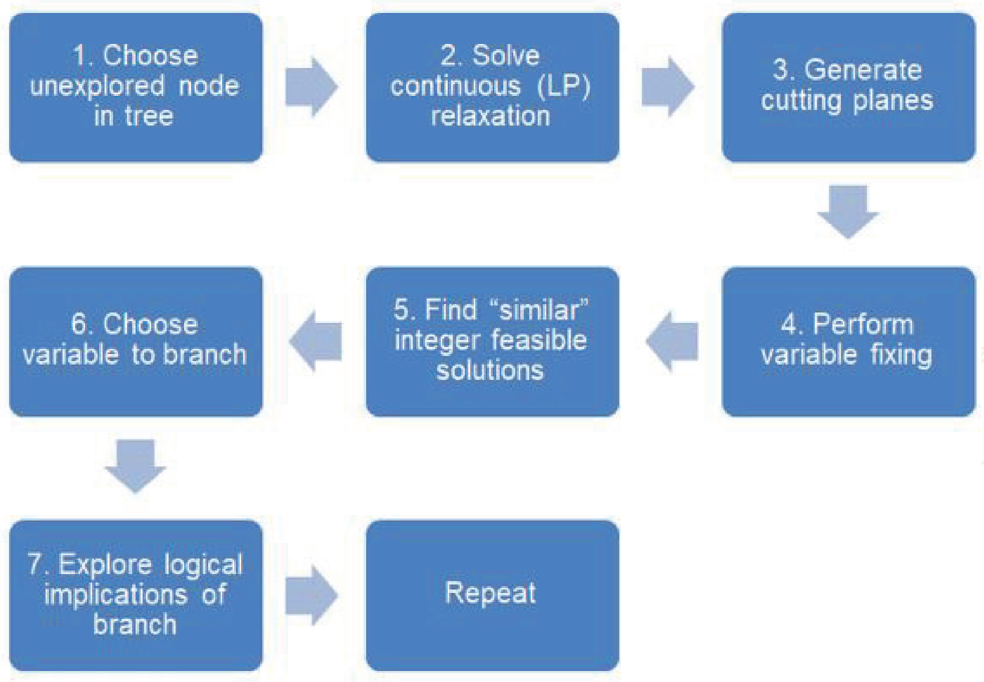
1. Choose unexplored node in tree
2. Solve continuous (LP) relaxation
3. Generate cutting planes
4. Perform variable fixing
5. Find "similar" integer feasible solutions
6. Choose variable to branch
7. Explore logical implications of branch
Repeat

Figure 2.
Key steps in the branch-and-bound procedure [9].

Typically, a few dual simplex iterations are sufficient to restore optimality, and the cost per node is quite small. The subsequent step 3 entails generating cutting planes as needed to obtain a continuous (LP) relaxation solution.

Step 4 involves variables fixing using reduced cost. If the following condition as given by Eq. (5) holds at a branch-and-bound node:

$$z_{LP} + \left|D_j\right| \geq z^* \tag{5}$$

where z_{LP} = objective value of LP relaxation solution at the root node, z^* = objective value of an incumbent (i.e., best known integer feasible solution), and D_j = reduced cost (marginal cost of releasing a variable from its bound), then we apply the strategy of fixing x_j to its current value in this subtree of the search. The goal here as described by step 5 is to obtain integer feasible solutions which are similar to the relaxation solution.

Selecting an appropriate branching variable can significantly affect the search tree size, which is emphasized in the subsequent step 6. In this regard, the guiding principles are to make the important decisions early (as modeled by the integral branching variables) by being aware of the impact of both branching directions. To illustrate by using a factory building problem, such a decision involves whether to build a factory first while the decision on the number of lines to be placed in the factory can be made later. In general, we can predict the impact of a branch by considering variables that are furthest from their bounds which indicate maximum infeasibility. Thus, the impact for each branching candidate can be measured to allow for strong branching to be performed, e.g., by using historical information such as pseudo-costs.

Finally, in step 7, the main idea in propagating implications logically is to fix the binary variables to possible values during tree exploration and determine the binary variable values. Bound strengthening is used to tighten variable bounds.

Practical considerations render implementing branch-and-bound to be unsuitable for large scale problems chiefly because the number of iterations grows exponentially with number of variables. Therefore in practice, a commercial business intelligence solver such as CPLEX uses a branch-and-cut procedure as a modification which applies model reformulation by using presolve strategies and adding cutting planes (or cuts) as shown in **Figure 3** with possible enhancement in practice around the root node computations [11].

3.3 Presolve and cutting planes

The original MIP formulation can be improved by tightening it with fewer constraints and variables thus entailing less data handling requirement (yet with the same solution quality). A tighter formulation also leads to a smaller difference between the space of the feasible continuous and feasible integer solutions, hence relying less on branching to refine the continuous relaxation computation. Two techniques are used: (1) presolve which combines preprocessing and probing strategies [12, 13]; and (2) cutting planes [14].

Presolve generates a new tighter improved model without size increase that is independent of the relaxation solution. Preprocessing aims to identify feasibility and redundancy while improving bounds (e.g., through rounding) while that of probing improves coefficients by fixing the binary variable values while checking for their logical implications. In both cases, we achieve a tighter model reformulation using similar steps of adding or replacing constraints that maintain the same integer solutions but with fewer continuous relaxation solutions. Adding a single constraint can produce an exponential number of tighter constraints. Such tighter constraints dominate the existing constraints without creating a larger problem. Note that reformulation solution is different from that of relaxation.

In contrast, we add a cutting plane (or valid inequality) to an existing model (typically the presolve-reformulated model) to remove a relaxation solution—this feature constitutes an important difference between the two techniques. Therefore, cutting planes introduce tighter constraints that cut off a particular relaxation solution and in so doing, achieves focused growth in model size.

In summary, presolve is vital in solving MIP as there is significant scope to improve most model formulations through reducing problem sizes (by more than

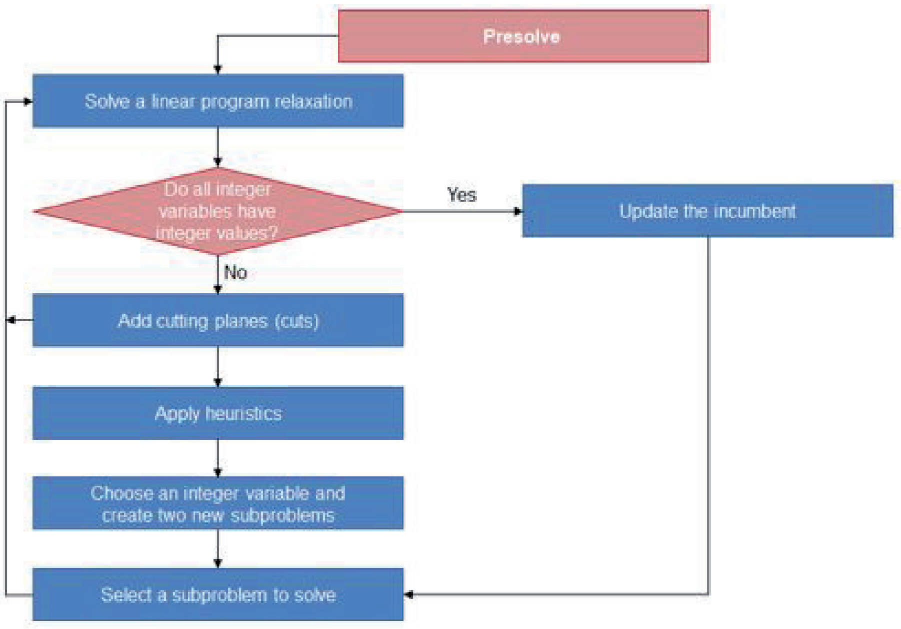
Presolve
Solve a linear program relaxation
Do all integer variables have integer values?
Yes
Update the incumbent
No
Add cutting planes (cuts)
Apply heuristics
Choose an integer variable and create two new subproblems
Select a subproblem to solve

Figure 3.
Branch and cut algorithm (CPLEX).

5 times is not uncommon) or runtimes (similarly by up to 10 times). On the other hand, cutting planes are available in numerous varieties with many valid types applicable for a particular model. Thus we need to identify relevant ones which serve to cut off appealing relaxation solutions. There is a need to strike a balance in terms of how many cuts to generate for a relaxation solution. Since we need to cut off relaxation solution only once, and it is expensive to resolve in obtaining a new relaxation solution for each cut added, we conduct multiple rounds of cutting plane generation while limiting the number of cuts per round in view of the increased model size [15].

3.4 Heuristics

Heuristics for solving MIP aims to produce good and possibly feasible solutions quickly without relying on branching in satisfying user demands for a problem. Thus, heuristics avoid exploring unproductive subtrees (in a branch-and-cut scheme) while exploring parts of tree that a solver typically will not. In doing so, heuristics help to prove optimality explicitly by pruning nodes more efficiently as well as implicitly by giving integer solutions [16].

Heuristics can be classified into two classes as available in a solver like CPLEX: (1) plunging (diving) heuristics, and (2) local improvement heuristics which explore interesting neighborhoods around potential solutions using search strategies such as local branching, relaxation induced neighborhood search (RINS), guided dives, and evolutionary algorithms for solution polishing. Plunging heuristics maintains linear feasibility in trying to achieve integer feasibility while local improvement heuristics operate conversely [17]. A typical strategy for heuristics applied at the root node involves the sequence shown in **Figure 4**.

Some considerations in applying plunging heuristics include tradeoffs of how many variables to fix per computation round and in what order. While it is computationally inexpensive to fix all variables rather than a few variables, LP relaxation solutions in the latter (not needed in the former) can guide later choices (e.g., on variable values and reduced costs). Variations in variable fixing order can be useful for diversification. On the other hand, a high-level structure of local improvement heuristics involves choosing integer values for all the integer variables, which produces linear infeasibility; iterating over the integer variables; and applying infeasibility metrics [16].

The effectiveness of heuristics is evidenced in that feasible solutions are found for most models before branch-and-bound is performed. Approximately 10% improvement in computational time to proven optimality has been reported [16]. Furthermore, heuristics often get solutions not obtained by branching.

3.5 Combined local search and heuristics

A combination of local search and heuristics offers a powerful optimization framework to solve difficult MIP or combinatorial optimization problems. Examples of local search methods include simulated annealing, tabu search, and genetic algorithms. Local search methods consist of the key strategies of neighborhood (i.e., considers a set of solutions in the vicinity of current solution); intensification (i.e., temporary focus on part of solution space), and diversification (i.e., mechanism to change focus occasionally). In applying local search to MIP, generally neighborhoods are based on the problem structure, e.g., nodes and edges in graphs with no high level structural information available in arbitrary MIP models [16]. A question that arises is how we can generate and explore an interesting

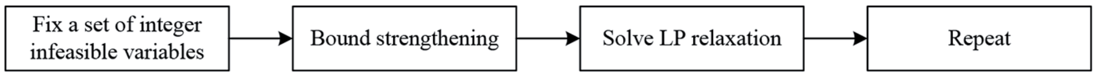

Figure 4.
Heuristics at root node.

neighborhood given an incumbent solution. In this regard, two methods are available, namely local branching [18] and relaxation induced neighborhood search (RINS) [19].

3.6 Parallelization

Parallelization is available in an integer optimization solver such as CPLEX, which encompasses the MIP solution engine, barrier algorithm, and concurrent optimization techniques for solving LP and QP problems. In the instance of CPLEX, parallelization involves launching several optimizers to solve the same problem—the process stops when the first solver reaches a solution. Within a branch and bound scheme, parallelization involves solution of the root node and nodes as well as strong branching in parallel [20].

3.7 Solution pools

The motivation to consider solution pools lies in the value of having more than one solution due to inaccurate data, approximations in model formulations, or inability of a model to capture the full essence of a problem. Thus, solution pools aim to generate and keep multiple solutions by using various options and tools that involve collecting solutions within a given percentage of optimal solution or those with diverse solutions and properties. However, difficulty is noted in implementing solution pools with the strategy of rolling horizon decompositions [17].

3.8 Tuning tools

As MIP solvers have multiple algorithm parameters which dictate their performance, the objective of tuning tool is to identify solver parameters that improve the performance for a given problem set. While default parameter values of MIP solvers are defined to work well for a large collection of problems, there is no such guarantee for a specific user problem [21].

4. Use cases

This section presents three use cases of applying commercial integer optimization solvers to implement and improve or enhance business intelligence applications. The model formulations for the use cases are implemented on GAMS modeling platform (and available in GAMS Model Library) from which the CPLEX solver is accessed.

4.1 Use case 1: energy optimization

The first use case presents a practical application of CPLEX as a standard solver for an energy business portfolio optimization problem for an electric utility company. For such electricity distribution public service, the problem involves to determine the amount to produce internally (i.e., in one's own power plant) and that to purchase externally (i.e., from the spot market or load following contracts). The problem formulation leads to a medium-to-large scale MILP model with size and computational statistics as described in **Table 3**. To accelerate solution convergence, several computational options are invoked including priority branching within a branch-and-bound procedure and multiple processing through parallelization (i.e., techniques introduced in the foregoing section). The computational results and implications as discussed in the cited reference demonstrates the applicability of the solver as an effective tool for 1-day ahead planning within a real-world electricity market in Germany.

4.2 Use case 2: financial optimization

The second use case involves financial optimization of risk management with commercial implications . The problem is amenable to be posed as an integer optimization model to capture an extensive set of rules and regulations that governs the delivery and settlement of mortgage-backed securities. The availability of reliable, robust, and efficient commercial integer optimization solvers alongside computing technology developments have facilitated the deployment and validation of such models with the computational statistics summarized in **Table 4**. The advancement achieved has led to optimization models including (if not particularly) integer programs to become essential omnipresent tools in current financial operations, which is comparable to the application of operations research and management science models in the domains of manufacturing, transportation, and logistics.

Computing platform	**GAMS 24.2.3 on laptop with Intel Core i7-8550U 1.80 (up to 1.99) GHz, 8 GB of RAM**
No. of continuous variables	1260
No. of discrete variables	773
No. of constraints	2178
No. of iterations	2,799,216
CPU time	408.234 second
Objective function value	EUR266,793 (for optimality gap = 0%)

Table 3.
Model size and computational statistics for use case 1.

Computing platform	**GAMS 24.2.3 on laptop with Intel Core i7-8550U 1.80 (up to 1.99) GHz, 8 GB of RAM**
No. of continuous variables	255
No. of discrete variables	199
No. of constraints	487
No. of iterations	238
CPU time	0.157 second
Objective function value	36.96 (for optimality gap = 1×10^{-4}%)

Table 4.
Model size and computational statistics for use case 2.

Computing platform	GAMS 24.2.3 on laptop with Intel Core i7-8550 U 1.80 (up to 1.99) GHz, 8 GB of RAM
No. of continuous variables	81
No. of discrete variables	8
No. of constraints	73
No. of iterations	15
CPU time	0.016 second
Objective function value	36.96 (for optimality gap = 0^4%)

Table 5.
Model size and computational statistics for use case 3.

4.3 Use case 3: manufacturing optimization

The third use case concerns production planning for a manufacturing facility . The application can be formulated as a standard integer optimization model of an uncapacitated lot-sizing problem. The objective function seeks to minimize production cost in meeting market demand constraints with cost components on production, stocking, and machine setups. **Table 5** gives the model size and computational statistics for the largest problem instance solved for this use case.

5. Conclusions

Performance variability across commercial integer optimization solvers applied to business intelligence applications (such as that for the use case in Section 4) occurs due to opportunistic parallelization, use of heuristics particularly by invoking polishing option (which involves random seed), or simply numerical reasons. Variability may be observed in computational time, performance in terms of number of nodes and iterations, or solution quality. A main limitation of the applicability of integer optimization solvers typically pertains to the number of integer variables that can be handled within acceptable computational load or solution time. Therefore, it is worthwhile for future research in this area to consider further improvement in the mentioned areas [22, 23] towards achieving acceptable performance levels that are requisite and crucial for business intelligence applications.

Acknowledgements

This work is completed partly under support from UTP-UCTS private grant no. 015MD0-037.

Author details

Cheng Seong Khor[1,2]

1 Chemical Engineering Department, Universiti Teknologi PETRONAS, Perak Darul Ridzuan, Malaysia

2 Centre for Process Systems Engineering, Institute of Autonomous Systems, Universiti Teknologi PETRONAS, Perak Darul Ridzuan, Malaysia

*Address all correspondence to: chengseong.khor@utp.edu.my; khorchengseong@gmail.com

References

[1] Tsay C, Baldea M. 110th anniversary: Using data to bridge the time and length scales of process systems. Industrial & Engineering Chemistry Research. 2019;**58**(36):16696-16708

[2] IBM. IBM ILOG CPLEX Optimization Studio V12.9.0. 2020. Available from: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html

[3] Gurobi Optimization. Gurobi Optimizer Reference Manual. Beaverton, Oregon: Gurobi Inc.; 2020

[4] Byrd RH, Nocedal J, Waltz RA. KNITRO: An integrated package for nonlinear optimization. In: Pillo GD, Roma M, editors. Large-Scale Nonlinear Optimization. Springer; 2006. pp. 35-59

[5] Jünger M et al. 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art. Berlin Heidelberg: Springer-Verlag; 2010. p. 804

[6] Rardin RL. Optimization in Operations Research. New Jersey: Prentice-Hall; 1998

[7] Williams HP. Model Building in Mathematical Programming. 4th ed. Chichester, West Sussex, England: John Wiley & Sons; 1999

[8] Danna E. Performance variability in mixed integer programming. In: Workshop on Mixed Integer Programming 2008 (MIP 2008). New York City, NY: Columbia University; 2008

[9] Rothberg E. The CPLEX library: Mixed integer programming. In: 4th Max-Planck Advanced Course on the Foundations of Computer Science (ADFOCS 2003). Saarbrücken, Germany: Max-Planck-Institut für Informatik; 2003

[10] Land AH, Doig AG. An automatic method of solving discrete programming problems. Econometrica. 1960;**28**(3):497-520

[11] Lima RM, Grossmann IE. On the solution of nonconvex cardinality Boolean quadratic programming problems: A computational study. Computational Optimization and Applications. 2017;**66**(1):1-37

[12] Savelsbergh MWP. Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal on Computing. 1994;**6**(4):445-454

[13] Wolsey LA. Integer Programming. Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester: Hoboken, NJ: Wiley; 1998. pp. 203-258

[14] Nemhauser G, Wolsey L. The theory of valid inequalities. In: Integer and Combinatorial Optimization. Hoboken, NJ: Wiley; 1988. pp. 203-258

[15] Rothberg E. The CPLEX library: Presolve and cutting planes. In: 4th Max-Planck Advanced Course on the Foundations of Computer Science (ADFOCS 2003). Saarbrücken, Germany: Max-Planck-Institut für Informatik; 2003

[16] Rothberg E. The CPLEX library: MIP heuristics. In: 4th Max-Planck Advanced Course on the Foundations of Computer Science (ADFOCS 2003). Saarbrücken, Germany: Max-Planck-Institut für Informatik; 2003

[17] Rothberg E. An evolutionary algorithm for polishing mixed integer programming solutions. INFORMS Journal on Computing. 2007;**19**(4):534-541

[18] Fischetti M, Lodi A. Local branching. Mathematical Programming. 2003;**98**(1):23-47

[19] Danna E, Rothberg E, Pape CL. Exploring relaxation induced neighborhoods to improve MIP solutions. Mathematical Programming. 2005;**102**(1):71-90

[20] Lima R. IBM ILOG CPLEX: What is inside of the box? In: Enterprise-Wide Optimization (EWO) Seminar. Pittsburgh, PA: Carnegie Mellon University; 2010

[21] IBM. CPLEX Performance Tuning for Mixed Integer Programs. 2019. Available from: https://www.ibm.com/support/pages/cplex-performance-tuning-mixed-integer-programs

[22] Bixby R et al. MIP: Theory and practice—Closing the gap. In: System Modelling and Optimization: Methods, Theory, and Applications. Boston, MA: Kluwer Academic Publishers; 2000. pp. 19-49

[23] Bixby R, Rothberg E. Progress in computational mixed integer programming—A look back from the other side of the tipping point. Annals of Operations Research. 2007;**149**(1):37-41