ENGINEERING Management 2026, 13(1): 1–16
https://doi.org/10.1007/s42524-026-5153-z

REVIEW ARTICLE

Lei HUANG, Fan XIAO, Dongdong GE, Wotao YIN, Zhe LIANG

An overview of mathematical programming solvers: Theory, development, and recent advances

Abstract Mathematical programming solvers are software tools designed to solve real-world problems using mathematical programming algorithms. This survey explores the evolution of optimization technologies, from traditional methods such as the simplex algorithm and branch-and-bound techniques to modern advancements that are facilitated by parallel computing, GPU acceleration, and AI algorithms. We also emphasize the recent emergence of mathematical programming solvers developed by research institutes and companies headquartered in China as major players, who have achieved remarkable success in benchmarks when compared to established solvers. This article provides a comprehensive overview of the theoretical foundations, historical progress, and emerging trends in mathematical programming solvers, offering valuable insights for both researchers and practitioners in the field.

Keywords survey, mathematical programming solver, linear programming, mixed-integer programming, nonlinear programming

Received Jan. 18, 2025; revised Aug. 6, 2025; accepted Aug. 23, 2025

Lei HUANG
School of Economics and Management, Tongji University, Shanghai 200092, China; College of Transport & Communications, Shanghai Maritime University, Shanghai 201306, China

Fan XIAO
SILC Business School, Shanghai University, Shanghai 200444, China

Dongdong GE
Antai College of Economics & Management, Shanghai Jiao Tong University, Shanghai 200030, China

Wotao YIN (✉)
DAMO Academy, Alibaba Group US, WA 98004, United States
E-mail: wotao.yin@alibaba-inc.com

Zhe LIANG (✉)
School of Economics and Management, Tongji University, Shanghai 200092, China
E-mail: liangzhe@tongji.edu.cn

This work was supported by the National Natural Science Foundation of China (Grant Nos. 72425001, 72401219, 72231006, and 72301165).

1 Introduction

Mathematical programming solvers are engineering software that employ mathematical programming algorithms to tackle large-scale, complex problems encountered by public institutions and commercial organizations. These technologies have been widely applied across various fields, including manufacturing, transportation, communication, finance, energy, supply chain, staff scheduling, logistics, and retail operations. Mathematical programming solvers significantly improve decision-making processes in the planning, assignment, and scheduling of limited resources.

Advanced mathematical programming solvers have dedicated considerable resources to addressing two fundamental issues in practical applications: linear programming (LP) and mixed-integer linear programming (MILP). However, real-world problems often extend beyond these, leading to the development of a variety of solvers that can handle more complex nonlinear problems, such as quadratic programming (QP), second-order cone programming (SOCP), semidefinite programming (SDP), and non-convex optimization problems. Further extensions that incorporate integer decision variables, such as mixed-integer quadratic programming (MIQP), mixed-integer second-order cone programming (MISOCP), and mixed-integer nonlinear programming (MINLP), have also attracted increasing attention in both research and solver development. The mathematical solvers are built upon a combination of well-established mathematical programming theories and algorithms, including the simplex method, interior-point methods, duality theory, branch-and-bound methods, first-order methods, active set methods, and various heuristic algorithms. More recently, the development of mathematical programming solvers has been experiencing a new wave of transformation driven by emerging technologies, such as the rise of parallel computing and artificial intelligence. These innovations have created new opportunities for enhancing solver performance and addressing increasingly complex problems.

The development of mathematical programming solvers faces considerable technical challenges. It requires not only expertise in mathematical optimization theory but also strong capabilities in software engineering, particularly in data structures, algorithm design, and lower-level implementation skills. Moreover, the development process is time-consuming, resource-intensive, and inherently risky. The historical advancement of these solvers has been developed mainly by prominent companies and institutions in Europe and the US, with notable commercial solvers like CPLEX, Gurobi, Mosek, Knitro, and Xpress. Benchmark test sets such as MIPLIB, created in 1992 (Bixby et al., 1992) and contributed by both companies and academic institutions, along with CBLIB (Friberg, 2016) and QPLIB (Furini et al., 2019), have been widely used for comparing the performance of optimizers and stimulating research in this area. Building on these libraries, Professor Hans Mittelmann at Arizona State University established a continuously maintained benchmarking platform that provides comparisons of solvers across LP, MILP, SDP, MIQP, MINLP and other problem classes. Although the benchmarks represent only a small fraction of real-world optimization problems, they help demonstrate the explosive speedups of solver performance in the last few decades.

In recent years, solvers developed by multinational companies headquartered in China have emerged as prominent contributors to the optimization field, making significant advancements in global optimization technology. Notable solvers include COPT, launched by Cardinal Operations in 2019; MindOpt, released by Alibaba in 2020; OptVerse and TAYLOR from Huawei in 2021 and 2024, respectively; and LeOPT from Lenovo in 2022. At present, some of these solvers have ranked among the top performers in the LP Simplex, Barrier, and MILP benchmarks maintained by Mittelmann. Although they entered the market relatively recently, their rise has been remarkably rapid.

Several reviews have already been conducted on mathematical programming solvers. Some studies focus exclusively on the CPLEX solver, covering the 1990s and 2000s (Achterberg and Wunderling, 2013; Bixby, 2002; Bixby et al., 2004), while others concentrate specifically on LP/MILP solvers, as discussed by Koch et al. (2022). Also, some reviews exclusively focus on MINLP solvers, such as Bussieck and Vigerske (2010). Nevertheless, most of these reviews do not address the emerging solvers of recent years. The purpose of this survey is to provide an overview of mathematical programming solvers, encompassing their theoretical foundations, historical development, and the latest advances, with coverage on newer techniques such as parallelization techniques, graphics processing unit (GPU) acceleration, and artificial intelligence (AI) algorithms.

The remainder of this survey is organized as follows. Section 2 presents the theoretical foundations of mathematical programming solvers, discussing the development of mathematical algorithms in optimization, focusing on linear programming, mixed-integer linear programming, and nonlinear programming. Section 3 provides a brief history of mathematical programming solvers and reviews the state-of-the-art solvers, with particular emphasis on leading Chinese solvers. Section 4 highlights recent advances, including parallel computing, GPU acceleration, and AI algorithms. Section 5 discusses the challenges in solver evaluation, especially in the context of real-world applications. Section 6 provides a conclusion to the paper.

2 Theoretical foundation

2.1 Linear programming

Linear programming deals with the problem of minimizing or maximizing a linear function in the presence of linear equality and/or inequality constraints, wherein the decision variables are continuous. The two most commonly used algorithms for linear programming in solvers are the simplex method and the interior point method, along with their variants. This section provides an overview of the foundational literature on these methods, comparing their theoretical principles and performance across different problem types and characteristics.

George B. Dantzig is widely recognized for pioneering linear programming, introducing the groundbreaking simplex method in 1947 (Dantzig, 2002), which became one of the most influential algorithms of the 20th century. Although Leonid V. Kantorovich had formulated similar problems and developed a solution method in 1939, his work remained unknown until 1959 (Bazaraa et al., 2010). Thus, the conception of the general class of linear programming problems is typically credited to Dantzig. In 1955, Dantzig and his colleagues formalized the theory behind the simplex method, establishing the concepts of basis construction, basic feasible solutions, boundedness, and finite termination (Dantzig et al., 1955). The principle behind the method relies on the fact that, under standard assumptions that the feasible region is a pointed polyhedron, an optimal solution (if it exists) to a linear program can be found at an extreme point (or vertex). Starting at an initial extreme point, the simplex method iteratively moves along the edges of the feasible region, transitioning from one extreme point to another until an optimal solution is found. In 1954, Lemke introduced the dual simplex method, which focuses on maintaining dual feasibility while addressing primal infeasibilities, providing an alternative approach for solving linear programs (Lemke, 1954) and proving particularly useful for re-optimization scenarios. Although the simplex method exhibits exponential worst-case complexity (Klee and Minty, 1972), average-case analysis suggests that the simplex method

can have polynomial average time complexity under certain assumptions on the input data (Borgwardt, 1982a, 1982b; Smale, 1983a, 1983b). Modern implementations further improve the efficiency of the simplex method through algorithmic refinements. For instance, most solvers compute and maintain an LU factorization rather than a basis inverse because the former is sparser and can be computed in a more numerically stable manner (Duff et al., 2017). To efficiently update this factorization during pivot steps, the Forrest–Tomlin update is commonly employed, which avoids recomputing the LU factors at each iteration by applying permutations and sparse triangular substitutions (Forrest and Tomlin, 1972).

Khachiyan (1979) introduced the ellipsoid method, demonstrating for the first time that linear programming problems can be solved in polynomial time. Nonetheless, despite its theoretical advancement, the ellipsoid method suffered from poor computational performance due to factors such as numerical instability and high per-iteration cost, which hindered its practical use. To the best of our knowledge, the ellipsoid method has not been implemented in any widely used commercial or open-source solvers. In 1984, Karmarkar made a significant advancement with the projective method, an interior-point algorithm that became the first practical polynomial-time method for linear programming problems (Karmarkar, 1984). Subsequent developments led to the primal-dual logarithmic barrier method, which, when combined with Mehrotra's predictor–corrector strategy (Mehrotra, 1992), became the method of choice in many modern solvers (Klotz and Newman, 2013). Unlike the simplex method, which moves along the boundary of the feasible region, interior-point methods progress through the interior and only as they converge to an optimal solution. Although different variants adopt different strategies for computing the search direction, they all require solving a linear system at each iteration, which is typically the most computationally intensive step. To improve efficiency, most solvers perform a symbolic Cholesky factorization of the system matrix in advance, exploiting its fixed sparsity pattern so that only the numerical values need to be updated during iterations (Klotz and Newman, 2013).

The simplex and interior-point algorithms perform differently depending on the structure of the linear programs. Lustig et al. (1994) demonstrated that carefully constructed interior point implementations can outperform simplex methods on large-scale problems. Terlaky (2009) reviewed the developments of interior point methods over 25 years, discussing their variants, advantages, and limitations. Klotz and Newman (2013) offered guidelines for choosing LP solving methods based on the structure of linear programming problems. For example, the aspect ratio n/m (the ratio of variables to constraints) is often considered a heuristic factor: the primal simplex algorithm may be preferred when $m \ll n$ or when the objective function changes frequently, while the dual simplex algorithm may perform better when $n \ll m$ or when constraints are added iteratively, as it maintains dual feasibility. However, such heuristics are not universally reliable, and no definitive condition based on m and n guarantees better performance of one method over another. Reflecting this uncertainty, most commercial linear programming solvers implement multiple algorithms, typically both primal and dual simplex, along with at least one interior-point method, and incorporate dynamic selection and switching strategies, sometimes switching multiple times within a single run to improve efficiency. Some solvers also employ parallel execution of different algorithms or by using a "crossover" strategy where an interior-point method is used to quickly find a near-optimal interior point, and then the simplex method is used to converge to an exact vertex solution if required.

2.2 Mixed-integer linear programming

A natural extension of linear programming is mixed-integer linear programming, in which some or all of the decision variables are restricted to discrete or integer values. While LP forms the theoretical foundation, MILP dominates real-world optimization problems due to its broad applicability (Bixby, 2012). The origins of MILP can be traced back to the classic paper by Dantzig, Fulkerson, and Johnson on the traveling salesman problem in 1954, frequently regarded as the birth of integer programming (Dantzig et al., 1954). In the past 70 years, MILP has developed into a dynamic field of research, characterized by significant theoretical advancements and practical applications. Most modern MILP solvers are based on the branch-and-bound method, which is introduced next.

The branch-and-bound method, first introduced by Markowitz and Manne (1957), uses intelligent enumeration to find an optimal solution for MILP problems. It builds a search tree, wherein each node represents a restricted version of the original problem with additional constraints to enforce integer solutions for specific variables. At each node, the algorithm solves a linear programming relaxation of the problem. Separately, cutting planes were pioneered by Dantzig et al. (1954, 1959) and Gomory (1958), introducing techniques to tighten the LP relaxation iteratively. In 1957, Markowitz and Manne integrated branch-and-bound with cutting planes, resulting in the branch-and-cut method (Markowitz and Manne, 1957). Two decades later, researchers successfully applied the branch-and-cut method to solve practical problems (Crowder et al., 1983; Van Roy and Wolsey, 1987). Subsequent research concentrated on refining the two main components of branch-and-cut: branching strategies and cutting planes.

Branching is essential to the exponential worst-case complexity of LP-based branch-and-cut algorithms. The strategy for splitting a problem into subproblems plays a

critical role in determining the size of the resulting search tree, therefore affecting the efficiency of solving the problem within a reasonable timeframe (Benichou et al., 1971). Common branching rules encompass most infeasible branching, which selects a variable with the least clear direction of rounding (Achterberg et al., 2005); pseudocost branching, where the coefficients and dual solution values of the rows are used as weights; and strong branching, which evaluates fractional candidates to identify the one that is likely to yield the most significant improvement in the solution (Applegate et al., 1995). More recently, hybrid branching methods have become increasingly popular. For instance, hybrid strong/pseudocost branching implements strong branching in the upper levels of the search tree (up to a predetermined depth), switching to pseudocost branching for deeper levels. Another approach, pseudocost with strong branching initialization, employs strong branching to variables with uninitialized pseudocosts as an initial estimate (Linderoth and Savelsbergh, 1999). Reliability branching takes this a step further by using strong branching not only on variables with uninitialized pseudocosts but also on those with unreliable pseudocost values (Achterberg et al., 2005). In addition, more sophisticated hybrid branching rules have been proposed to further refine the branching process (Achterberg and Berthold, 2009).

Cutting planes refine LP relaxations by removing infeasible regions without excluding feasible integer solutions. One of the most classic cutting plane algorithms is Gomory's mixed-integer cuts (Gomory, 1960), a generalization of his fractional cuts (Gomory, 1958), which is designed for general mixed-integer programming without exploiting specific problem structure. Since then, various other types of cuts have been proposed, such as disjunctive cuts (Balas, 1979), knapsack cover cuts (Johnson and Padberg, 1982; Weismantel, 1997), flow path cuts (Van Roy and Wolsey, 1985), implied bound cuts (Hoffman and Padberg, 1991), lift and project cuts (Balas et al., 1993), GUB cover cuts (Gu et al., 1998), flow cover cuts (Padberg et al., 1985; Gu et al., 1999), clique cuts (Johnson and Padberg, 1982; Atamtürk et al., 2000), mixed-integer rounding cuts (Marchand and Wolsey, 2001), and multi-commodity network flow cuts (Achterberg and Raack, 2010). Bixby and Rothberg (2007) noted that evaluating new cutting plane methods in isolation is challenging due to their complex interactions. To address this, the study deactivated one method at a time from several cuts to assess the impact of each. The results revealed that removing Gomory's mixed-integer cut, despite being the oldest, caused the greatest performance degradation, underscoring its effectiveness. Additionally, while many cutting plane methods are broadly effective across a wide range of problem instances, some are more specialized and perform better for specific types of models (Achterberg and Wunderling, 2013).

In addition to the branch-and-cut method introduced earlier, two widely used techniques in MILP solvers for accelerating the solution process, presolving and primal heuristics, are presented next.

Presolving, or preprocessing, is essential for the efficient resolution of integer programs, as emphasized by Brearley et al. (1973) and Savelsbergh (1994). Its goal is to quickly identify and eliminate redundant constraints and variables, as well as tighten the bounds, resulting in a smaller and more solvable problem. This is particularly important in branch-and-bound methods, which often require solving numerous linear programming problems (Wolsey, 2020). Presolving techniques can be applied both at the root node, prior to solving the first linear relaxation, and at subsequent nodes in the search tree (Achterberg et al., 2020). Studies have demonstrated that presolving substantially enhances the performance of mixed-integer programming solvers (Achterberg et al., 2020; Achterberg and Wunderling, 2013; Bixby et al., 2004).

Primal heuristics aim to quickly find good feasible solutions, which helps reduce the search tree size in branch-and-bound algorithms by enabling earlier pruning of subtrees. In practice, providing a good solution is often sufficient, as proving optimality may be computationally infeasible (Achterberg and Wunderling, 2013). These heuristics are typically classified into two categories: starting heuristics, or construction heuristics, which generate an initial feasible solution, and improvement heuristics, which refine existing solutions. Examples of starting heuristics include dive-and-fix (Berthold, 2006; Wolsey, 2020), neighborhood rounding (Berthold, 2006; Wolsey, 2020), relaxation-enforced neighborhood search (Berthold, 2006), and feasibility pump (Fischetti et al., 2005). Examples of improvement heuristics include local branching (Fischetti and Lodi, 2003), proximity search (Fischetti and Monaci, 2014), RINS (Danna et al., 2005), and polishing (Rothberg, 2007). Starting and improvement heuristics are mutually dependent: starting heuristics provide the initial solutions required by improvement heuristics for refinement, while improvement heuristics enhance the often low-quality solutions generated by starting heuristics (Achterberg and Wunderling, 2013).

2.3 Nonlinear programming

In practice, many problems are nonlinear, in addition to the aforementioned LP/MILP models. This encompasses a broad range of nonlinear optimization, including quadratic programming, second-order cone programming, semidefinite programming, and general nonlinear programming (NLP). For problems involving discrete variables, these problems become mixed-integer nonlinear programming problems, which inherit the combinatorial complexity of integer programming together with nonlinear optimization.

Quadratic programming refers to optimization problems

characterized by a quadratic objective function and linear equality or inequality constraints. QP with linear constraints can be regarded as a generalization of linear programming, in which the objective function is quadratic. When the constraints include quadratic terms, the problem is referred to as quadratically constrained quadratic programming (QCQP). QP problems are NP-hard (Floudas and Visweswaran, 1995). However, if the matrix in the quadratic terms in the objective function is positive semidefinite or positive definite, the problem is convex and can be solved in polynomial time. Convex QP has been studied since the 1950s (Frank and Wolfe, 1956), and efficient solution methods have been developed over time. Common approaches for solving convex QP problems include active-set methods (Wolfe, 1959), interior-point methods (Wright, 1997), and first-order methods such as the alternating direction method of multipliers (ADMM) (Boyd, 2010). Among these, interior-point methods are typically the default algorithms in current commercial solvers.

Second-order cone programming involves minimizing a linear function over the intersection of an affine set and the Cartesian product of second-order cones (Nesterov and Nemirovskii, 1994). SOCP problems are nonlinear convex programming problems that are extensions of both linear programming problems and convex quadratic programming problems, but are less general than SDP problems. In particular, convex quadratically constrained quadratic programs can be transformed into equivalent SOCP problems through certain reformulations. SOCP problems have a wide range of applications, including antenna array weight design, grasping force optimization, finite impulse response filter design, portfolio optimization with loss-risk constraints, SOC-representable functions and sets, and robust linear programming (Lobo et al., 1998). The primal-dual interior-point method, first developed by Kojima et al. (1989) and Monteiro and Adler (1989), is currently the core algorithm for solving SOCP problems.

Semidefinite programming involves minimizing a linear function subject to the constraint that an affine combination of symmetric matrices is positive semidefinite. SDP is a subclass of convex optimization due to the fact that semidefinite constraints are convex, despite their nonlinear and non-smooth nature (Vandenberghe and Boyd, 1996; Gärtner and Matoušek, 2012). SDP serves as a standard formulation that generalizes linear programming, convex quadratic programming, and second-order cone programming (Alizadeh and Goldfarb, 2003). Several methods have been developed for solving SDP problems, including the ellipsoid method (Gärtner and Matoušek, 2012), interior-point methods (Vandenberghe and Boyd, 1996), first-order methods (Wen et al., 2010), and approximate algorithms (Gärtner and Matoušek, 2012). Among these, interior-point methods are the most widely adopted algorithms due to their robustness and efficiency in solving general linear SDP problems, forming the foundation of most modern solvers.

General nonlinear programming problems play a critical role in many industrial contexts, such as process optimization in the oil and chemical industries, nonlinear network flow problems in electric power, water, and gas distribution systems, urban traffic equilibrium models, and economic planning models (Lasdon and Waren, 1980). Two major solution approaches are sequential quadratic programming and interior-point methods. Sequential quadratic programming, first proposed by Wilson (1963), solves a sequence of QP subproblems that approximate the original problem by minimizing a quadratic model of the objective function subject to linearized constraints (Nocedal and Wright, 2006). Interior-point methods, by contrast, tackle the primal-dual optimality conditions directly through Newton-type algorithms applied to relaxed barrier formulations. In addition, researchers have also proposed strategies to address large-scale or structured NLP problems, such as gradient projection methods (Kelley, 1999), linearly constrained augmented Lagrangian methods (Gill et al., 2019), and difference-of-convex (DC) programming approaches (Tao and An, 1997). Specifically, DC programming reformulates nonconvex objectives as the difference of two convex functions. A variety of algorithms have been developed to solve DC programs, including the DC algorithm (DCA) (An and Tao, 2005), the convex-concave procedure (CCP) (Yuille and Rangarajan, 2003), and several proximal or penalized variants (Lipp and Boyd, 2016; Wen et al., 2018) designed to enhance convergence properties.

Similar to MILP, many practical problems in nonlinear programming involve integer variables, leading to MINLP problems. Depending on the structure of the problem, the aforementioned nonlinear programming problems become MIQP/MIQCQP, MISOCP, MISDP, or generalized MINLP, depending on the problem structure. Applications of MINLP include options pricing, portfolio optimization, network design and operations, statistics, scheduling, and logistics (Benson and Sağlam, 2013). MINLP problems can be classified into convex and nonconvex problems, based on whether the relaxation of the integer constraints results in a convex problem (Burer and Letchford, 2012; D'Ambrosio and Lodi, 2013). For convex MINLP, various solution methods have been proposed, including generalized Benders' decomposition (Geoffrion, 1972), branch-and-bound (Gupta and Ravindran, 1985), outer approximation (Duran and Grossmann, 1986), LP/NLP-based branch-and-bound (Quesada and Grossmann, 1992), the extended cutting-plane method (Westerlund and Pettersson, 1995), branch-and-cut (Stubbs and Mehrotra, 1999), and hybrid approaches (Bonami et al., 2008; Abhishek et al., 2010). For nonconvex MINLP, the continuous relaxation of the problem is a global optimization problem and is NP-hard (Sherali

and Adams, 2013). Algorithms for solving non-convex MINLP include spatial branch-and-bound (Vigerske and Gleixner, 2018), branch-and-reduce (Ryoo and Sahinidis, 1995), and α-branch-and-bound (Androulakis et al., 1995). These techniques form the algorithmic foundations of deterministic global optimization solvers such as BARON, ANTIGONE, and Couenne (D'Ambrosio and Lodi, 2013).

Beyond the above deterministic methods, which typically rely on gradient information, a distinct class of derivative-free optimization (DFO) algorithms has been developed for problems where derivatives are unavailable, unreliable, or prohibitively expensive to obtain (Rios and Sahinidis, 2013). These include local methods, such as direct search (e.g., the Nelder–Mead simplex method (Lagarias et al., 1998)) and trust-region interpolation methods (Conn et al., 2000), as well as global approaches, such as response surface methods (Khuri and Mukhopadhyay, 2010) and heuristic methods such as simulated annealing (Metropolis et al., 1953). While DFO methods generally lack rigorous global optimality guarantees, they have been widely applied in simulation-based optimization, engineering design, hyperparameter tuning, and facility location problems.

3 History and development of mathematical programming solvers

The first systematic development of computer codes for solving linear programming problems using the simplex algorithm was carried out by Orchard-Hays and Dantzig at RAND Corporation. Their initial implementation used a device known as a Card Programmable Calculator (Orchard-Hays, 1990), which, although not a true computer, was a programmable calculator used for executing complex calculations. Between 1954 and 1956, these algorithms were successfully implemented on scientific computers such as the IBM 701 and IBM 704. Subsequently, Orchard-Hays moved from RAND to the CEIR, where he developed the LP/90/94 codes along with his colleagues. LP/90/94 was a milestone in optimization because it became, by all accounts, the first commercially used mixed-integer programming code based on the branch-and-bound method. In the 1980s, the emergence of modern mathematical programming solvers, such as LINDO, Xpress, and CPLEX, marked a significant advancement in the field of optimization (Bixby, 2012).

Building on this historical background, the remainder of this section will briefly introduce several representative modern mathematical programming solvers, summarize their licensing models and the types of problems they are designed to address, and discuss the recent development of China-originated mathematical programming solvers.

3.1 Category of modern mathematical programming solvers

Mathematical programming solvers can be broadly classified into two principal categories based on licensing: commercial and open-source. We present a selection of widely used solvers in these two categories: Table 1 lists commercial solvers (including those offering academic licenses), and those of open-source alternatives. Each table summarizes the solver's developer, release year, license type, and supported problem domains, including LP, MILP, QP/MIQP, QCQP/MIQCQP, SOCP/MISOCP, SDP/MISDP, NLP/MINLP, and constraint programming (CP).

The well-known commercial solvers from countries in Europe and the US include CPLEX (IBM ILOG, 2022), Gurobi (Gurobi Optimization, 2025), and Xpress (FICO, 2025). A brief overview of these three solvers is provided below.

CPLEX is a widely used mathematical programming solver that was originally developed by Dr. Robert Bixby in the 1980s under CPLEX Optimization Inc. The first version, CPLEX 1.0, was released in 1988. In 2009, CPLEX was acquired by IBM, which incorporated it into its IBM ILOG CPLEX Optimization Studio. From version 1.0 to 7.1, CPLEX achieves over a 50-fold speed improvement, with an overall speedup of 103 times due to both algorithm optimization and machine performance enhancements (Bixby, 2002). The latest version, CPLEX 22.1.2, released in 2022, supports a broad range of optimization problem types, including linear, mixed-integer, quadratic (both convex and non-convex), second-order cone programming, and constraint programming.

Zonghao Gu, Edward Rothberg, and Robert Bixby founded Gurobi in 2008, deriving the name by combining the first two initials of their last names. In 2009, the first version of the Gurobi solver, Gurobi 1.0, was released. According to benchmark tests maintained by Mittelmann, the performance of this version was roughly comparable to CPLEX 11.0 (Bixby, 2012). Today, Gurobi supports a wide range of optimization problems, including linear programming, quadratic programming, quadratically constrained quadratic programming, mixed-integer linear programming, mixed-integer quadratic programming, and mixed-integer quadratically constrained quadratic programming (Gurobi Optimization, 2025). The latest version, Gurobi 12.0, was released in November 2024.

Xpress was originally developed by Dash Optimization, with the initial authors being Bob Daniel and Robert Ashford. The solver was first released in 1983 as a tool for modeling and solving linear programming problems, and was later extended in 1986 to handle mixed-integer programming models (Ashford, 2007). In 2008, Xpress was acquired by FICO. Xpress is capable of solving a wide range of optimization problems, including linear, integer, quadratic, nonlinear, and stochastic programming

Table 1 Comparison of optimization solvers and their supported problem types

Solver	Developer	First release	License type[a)]	Problem domain												
				LP	MILP	QP	MIQP	QCP	MIQCP	SOCP	MISOCP	SDP	MISDP	NLP	MINLP	CP
LINDO (LINDO Systems, 2025)	LINDO Systems	1979	Proprietary/Academic-free	√	√	√	√	√		√		√		√	√	
Xpress (FICO, 2025)	FICO	1983	Proprietary/Academic-free	√	√	√	√	√	√	√	√			√	√	√
CPLEX (IBM ILOG, 2022)	IBM	1988	Proprietary/Academic-free	√	√	√	√	√	√	√	√					√
MOSEK (MOSEK ApS, 2025)	MOSEK ApS	1999	Proprietary/Academic-free	√	√	√	√	√	√	√	√	√				
BARON (The Optimization Firm, 2025)	The Optimization Firm	2001	Proprietary/Academic (discounted)	√	√	√	√	√	√	√	√	√	√	√	√	
KNITRO (Artelys, 2024)	Artelys	2001	Proprietary/Academic-free	√	√	√	√	√	√	√				√		
Gurobi (Gurobi Optimization, 2025)	Gurobi	2008	Proprietary/Academic-free	√	√	√	√	√	√	√	√			√	√	
ANTIGONE (Misener and Floudas, 2014)	GAMS	2014	Proprietary					√	√					√	√	
COPT (Ge et al., 2023)	Cardinal Operations	2019	Proprietary	√	√	√	√	√	√	√	√	√		√		
MindOpt (Alibaba DAMO Academy, 2025)	Alibaba DAMO Academy	2020	Proprietary	√	√	√	√	√	√			√		√		
OptVerse (Huawei, 2025)	Huawei	2021	Proprietary	√	√	√		√		√						
LeOPT (Lenovo, 2024)	Lenovo	2022	Proprietary	√	√											
TAYLOR (Huawei Taylor Lab, 2024)	Huawei Taylor Lab	2024	Proprietary	√	√	√								√		
XOPT (SRIBD, 2024)	SRIBD	2024	Proprietary	√	√											
Clp (COIN-OR Foundation, 2025a)	COIN-OR	2003	Open-source	√		√										
Cbc (COIN-OR Foundation, 2025b)	COIN-OR	2005	Open-source		√		√									
SCIP (Bolusani et al., 2024)	ZIB	2005	Open-source	√	√		√		√		√		√		√	
Couenne (Belotti, 2009)	COIN-OR	2009	Open-source												√	
HiGHS (Huangfu and Hall, 2018)	HiGHS	2020	Open-source	√	√	√										

Note: a) "Academic-free" refers to free full-featured academic licenses. "Academic (discounted)" refers to licenses offered to academic users at reduced cost. Trial licenses and community editions are not included in this classification.

(Laundy et al., 2009). The latest version, Xpress 9.6, was released in April 2025 (FICO, 2025). Additionally, Xpress includes Knitro, a specialized solver designed for solving CP problems.

In addition to commercial solvers, the open-source ecosystem has played a crucial role in advancing optimization technology. COIN-OR (Computational Infrastructure for Operations Research) is a prominent initiative launched by IBM Research in 2000 and established as a non-profit foundation in 2004 (Lougee-Heimer, 2003). It encompasses over 60 projects, each focusing on computational optimization, including Clp for linear programming, Cbc for integer programming, and Couenne for mixed-integer nonlinear programming, among others. Separately, HiGHS (Huangfu and Hall, 2018) is recognized as a strong open-source solver for linear and mixed-integer linear problems, with solid performance on benchmark tests such as those maintained by Mittelmann. Another leading open-source solver is SCIP (Bolusani et al., 2024), developed at Zuse Institute Berlin. First released in 2005, SCIP is one of the fastest open-source solvers for mixed-integer programming problems, and provides a powerful framework for constraint integer programming and branch-cut-and-price techniques.

3.2 Development of China-originated mathematical programming solvers

In recent years, research teams developed by research institutes and multi-national companies headquartered in China have introduced their independently developed solvers. In 2017, the Research Institute for Interdisciplinary Sciences at Shanghai University of Finance and Economics released LEAVES, China's first open-source optimization solver platform, supporting LP, SDP, geometric programming, and other convex problems. In 2018, the Academy of Mathematics and Systems Science at the Chinese Academy of Sciences released CMIP 1.0, the first mixed-integer programming solver developed domestically in China. A notable breakthrough occurred around 2020 with the emergence of a series of high-performance commercial solvers. These include COPT by Cardinal Operations (first released in 2019); MindOpt by Alibaba DAMO Academy (first released in 2020); OptVerse and TAYLOR by Huawei (first released in 2021 and 2024, respectively); and XOPT by the Shenzhen Research Institute of Big Data (first released in 2024). In this context, we will briefly introduce COPT, MindOpt, OptVerse, and TAYLOR, since they have demonstrated superior performance relative to other solvers.

In 2019, Cardinal Operations launched COPT, the first Chinese commercial linear programming solver. The initial version, COPT 1.0, was released on May 28, 2019. Subsequent versions introduced expanded capabilities: COPT 2.0, released in May 2021, added support for integer programming, and COPT 3.0, launched in October 2021, became the first Chinese commercial solver for second-order cone programming. In 2022, COPT underwent further advancements with COPT 4.0 (February), which included support for convex quadratic programming and convex quadratic constrained programming, followed by COPT 5.0 (June) with a semi-definite programming solver. By November 2022, the solver also included mixed-integer second-order cone programming. Today, COPT supports a comprehensive range of problem types, including linear programming, mixed-integer linear programming, quadratic and convex quadratic programming, semi-definite programming, general nonlinear programming, and various mixed-integer programming models, such as mixed-integer second-order cone programming, mixed-integer convex quadratic programming, and mixed-integer convex quadratic constrained programming.

In 2020, Alibaba DAMO Academy's Decision Intelligence Laboratory introduced the commercial solver MindOpt. The first public version, MindOpt Solver 0.9.0, was released on August 14, 2020, featuring a simplex method for solving linear programming problems. In May 2022, version 0.19.0 introduced support for convex quadratic programming, followed by version 0.20.0 in August 2022, which added support for mixed-integer linear programming. By November 2022, version 0.23.0 included semi-definite programming capabilities. In October 2023, MindOpt Solver 1.0 was launched, marking a significant milestone in its development. The latest version, MindOpt Solver 2.0.0 was released in November 2024 with new features that include support for quadratic constrained programming, mixed-integer quadratic programming, and mixed-integer quadratic constrained programming. Besides the classic solvers, MindOpt also features a black-box tuner (Zhang et al., 2023b), which can be used to optimize the performance of MILP solvers (Sun et al., 2024), as well as a development platform, MindOpt Studio, which encompasses optimization modeling, distributed computing, and all phases of full-stack development.

In September 2021, Huawei announced the release of OptVerse Optimizer at the Huawei Connect 2021 conference. The current version, 1.0.0, is available exclusively on Huawei Cloud. It supports solving large-scale linear programming, quadratic programming, and mixed-integer linear programming problems. Additionally, the solver offers support for a range of algorithmic models, including linear, nonlinear, mixed-integer, and quadratic constrained programming. In 2024, Huawei Taylor Laboratory proposed the TAYLOR solver, which is capable of solving large-scale mixed-integer programming, linear programming, quadratic programming, and nonlinear programming problems.

COPT, MindOpt, and OptVerse all achieved first place on Mittelmann's LP benchmark at the time of their initial releases. Subsequent benchmark snapshots confirm the

strong performance of China-originated solvers. For example, in the most recent results, COPT and OptVerse rank first and second on both the LPfeas and LPopt benchmarks—representing barrier and simplex methods, respectively—while COPT, OptVerse, and TAYLOR hold the top three positions on the MIPLIB2017 benchmark. In the domain of nonlinear optimization, COPT currently ranks first across several categories, including SDP, SOCP, MISOCP, AMPL-NLP, and QPLIB. OptVerse and TAYLOR also perform strongly, ranking second on the SOCP and AMPL-NLP benchmarks, respectively. Although MindOpt withdrew from public benchmarking in December 2024, its earlier results remain indicative of strong performance. For instance, it ranked third on both LPfeas and LPopt (October 28, 2024) and second on AMPL-NLP and QPLIB (October 28 and October 9, 2024, respectively).

While the recent benchmark results suggest strong performance from these solvers, some practical considerations should be noted. The rankings are based on tests conducted with default solver settings and wall-clock runtimes on specific hardware: LP benchmarks were run on an Intel i7-11700K (3.6GHz, 64GB RAM, Linux), while MIPLIB2017 was tested on an AMD Ryzen 9 5900X (12 cores, 128GB RAM, Linux). In real-world applications, solver performance may vary significantly depending on problem structure, tuning, and execution environment. Additionally, none of the above-mentioned Chinese commercial solvers currently offer free academic licenses, which may limit their accessibility for research and education.

In addition to commercial solvers, several open-source solvers have been proposed by research teams involving Chinese researchers, particularly GPU-based solvers. For instance, Lu et al. (2024) introduced cuPDLP-C, a GPU-based LP solver, while Han et al. (2024) proposed cuLoRADS, a GPU-based SDP solver. Both solvers achieved top rankings on Mittelmann's LP benchmark, tested on NVIDIA RTX A6000 GPUs. Although GPU-based solvers generally have limited numerical precision (typically with tolerances around 1e-4), these results demonstrate their considerable potential in practice, particularly when equipped with higher-performance GPUs or applied to relatively well-conditioned problems.

4 Recent advances

In practical applications, the performance of mathematical programming solvers is significantly influenced by advancements in computer technology. Their underlying theory and implementation techniques have also evolved alongside these technological developments. Some solvers have been restructured to optimize performance on modern high-performance computing architectures, while others have incorporated algorithms from diverse domains—particularly artificial intelligence—by integrating methods from machine learning, deep learning, and reinforcement learning to improve solver capabilities and computational efficiency. This section focuses on the influence of parallel computing, GPU acceleration, and AI algorithms on the advancement of mathematical programming solvers.

4.1 Parallel computing

Parallelization has become a crucial approach in accelerating solving optimization problems, particularly with the pervasiveness of multi-core CPU architectures. Modern MILP solvers are optimized for shared-memory parallelism on multi-core systems, enhancing computational efficiency. Benchmarks on data sets like MIPLIB 2017 have been conducted by Mittelmann and others on very capable hardware like the AMD Ryzen 9 5900X (12 cores, 128GB). Beyond shared-memory environments, distributed-memory parallelism—leveraging grid computing, supercomputers, and large-scale distributed systems—has been developed. For example, ParaSCIP has utilized up to 7,000 and 80,000 cores on the HLRN II and Titan supercomputers, respectively, solving multiple previously open instances from MIPLIB2010 within a two-hour time limit (Shinano et al., 2016).

Parallelization techniques differ on the type of problem. For LP problems, efforts have mainly concentrated on optimizing the interior point (barrier) method (Andersen and Andersen, 2000; Gondzio and Sarkissian, 2003). For MIP problems, state-of-the-art solvers based on the branch-and-cut algorithm utilize parallelization strategies that vary in granularity. These range from tree parallelism, which searches multiple trees in parallel, to subtree parallelism, which explores multiple subtrees simultaneously. Finer-grained approaches include node parallelism, where multiple nodes are processed concurrently, and subnode parallelism, which parallelizes tasks within a single node. Subnode parallelism techniques include strong branching, solving LPs, applying heuristics, generating cutting planes, and decomposition (Ralphs et al., 2018).

Although shared-memory parallelism significantly enhances solution speed within the capabilities of a single computing node, the memory and core count eventually become limiting factors. Distributed-memory parallelism addresses this by exploiting massive computational resources across multiple computing nodes, substantially improving the solvability of extremely large and difficult instances. A notable example is ParaSCIP, which successfully solved 14 previously unsolved MILP instances from the MIPLIB2003 and MIPLIB2010 benchmark sets within a two-hour time limit by utilizing up to 80,000 cores on large-scale supercomputing platforms (Shinano et al., 2016).

4.2 GPU acceleration

Over the past 10 to 15 years, GPUs have emerged as an essential element in scientific computing (Dongarra, 2022). A GPU consists of multiple cores, each functioning as a multithreaded SIMT (Single Instruction, Multiple Threads) processor. Compared to CPUs, GPUs exhibit two primary distinctions. First, GPUs possess a substantially greater number of cores. Most CPUs possess between 2 and 64 cores (though AMD EPYC 9005 is equipped with 192 cores), whereas GPUs sometimes contain thousands or more, allowing them to manage extensive simultaneous tasks. Second, GPUs operate on a SIMT architecture, where multiple threads execute the same instruction on different data elements. Additionally, GPUs employ a shared memory architecture within thread blocks, complemented by global memory accessible to all cores. These characteristics make GPUs highly efficient for data-parallel tasks, such as matrix operations and simulations, although CPUs remain superior for tasks requiring complex control logic.

Even though GPUs have attracted significant attention from solver developers, the majority of mainstream commercial solvers have yet to utilize them. For example, Gurobi states that GPUs are not well suited for LP/MIP/QP solvers for two main reasons: first, GPUs struggle with sparse linear algebra, which is central to linear programming; second, GPUs are optimized for SIMT operations, while parallel MIP requires different computations at each search tree node (Gurobi Optimization, 2023). Nevertheless, some researchers have explored the potential of GPUs for mathematical programming solvers, with promising results in certain contexts, as outlined below.

In linear programming, researchers have identified specific strategies to overcome the challenges associated with GPU implementation. Lu and Yang (2024b) highlighted sparse linear algebra as a key bottleneck in traditional LP solvers, such as those based on the simplex or barrier methods. To address this issue, they propose cuPDLP, a GPU-accelerated implementation of the restarted primal-dual hybrid gradient (PDHG) method, which is better suited for parallel computation on GPUs. They also introduce the restarted Halpern primal-dual hybrid gradient (rHPDHG), along with an extended version r^{2}HPDHG. Preliminary tests on LP relaxations from the MIPLIB data set show that cuPDLP exhibits promising performance on certain large-scale instances, and that r^{2}HPDHG can outperform traditional solvers in specific problem settings. In a follow-up study, Lu et al. (2024) re-implemented cuPDLP in C (cuPDLP-C), achieving a 50% speed improvement over the original Julia version. Moreover, cuPDLP-C exhibits distinct advantages over traditional interior-point and simplex solvers for extremely large-scale LP problems.

In the domain of nonlinear programming, Schubiger et al. (2020) proposed using GPUs to accelerate the ADMM method for large-scale quadratic programming. Similarly, Lu and Yang (2024a) introduced a first-order method known as the restarted accelerated primal-dual hybrid gradient (rAPDHG) for solving convex quadratic programming problems and presented the results on instances from QPLIB. Huang et al. (2024) further developed a restarted primal-dual hybrid conjugate gradient (PDHCG) method based on this framework, achieving up to 5 times speedup over rAPDHG and nearly 100 times over other solvers in specific large-scale tests. For semidefinite programming, Han et al. (2024) proposed GPU-accelerated low-rank factorization methods and introduced cuLoRADS, a GPU-based solver based on the Low-Rank ADMM Splitting approach. It solves MaxCut instances with $10^7 \times 10^7$ matrix variables in 10–60 s on an NVIDIA H100 GPU, whereas earlier CPU-based solvers required at least dozens of hours per instance.

For mixed-integer linear programming, however, GPU acceleration remains less feasible. The parallel workloads in MILP solvers, particularly the LP subproblems, can vary significantly, making it difficult to leverage GPUs effectively, as each node in the search tree may require different computations (Rothberg, 2020).

4.3 AI algorithms

Artificial intelligence, encompassing techniques including machine learning, deep learning, and reinforcement learning, excels at extracting patterns from vast amounts of historical data. This capability has spurred significant interest in leveraging AI to enhance optimization processes. In particular, the integration of AI with combinatorial optimization has emerged as a prominent research area, aiming to accelerate traditional solution methods and address complex, large-scale problems more efficiently.

AI algorithms have recently been applied to solve optimization problems, including mixed-integer programming. Some approaches use end-to-end learning, where models are trained to directly generate solutions from input data, applied successfully to problems like the traveling salesman and vehicle routing (Bengio et al., 2021). Others integrate AI with heuristic methods, such as large neighborhood search and the feasibility pump, to improve search efficiency and solution quality (Zhang et al., 2023a). However, while these methods offer promising results, their solution quality can be inconsistent, and they are not easily incorporated into traditional solvers. A more promising direction involves combining AI with exact methods like branch-and-cut to enhance solution efficiency (Bengio et al., 2021; Zhang et al., 2023a). This approach, which focuses on improving the efficiency of exact algorithms through AI, holds the greatest potential for practical integration into mathematical programming solvers.

The integration of AI algorithms with the branch-and-cut method focuses on key components such as branching variable selection, cutting plane selection, and advanced subroutines like primal heuristics. Branching variable selection is crucial due to the exponential growth of the search tree, where poor decisions can significantly increase its size. A prominent approach is to learn branching policies by imitating strong branching rules. For instance, Nair et al. (2021) used a bipartite graph representation of MILP problems, proposing a neural branching method that reduced the objective value gap with a smaller search tree and showed notable improvement over SCIP. In cutting plane selection, AI models are trained to identify promising cutting planes, which can tighten the linear programming relaxation and thus solve the problem fast. For example, Tang et al. (2020) introduced a Markov decision process formulation and trained a reinforcement learning agent to improve the selection of cutting planes, enhancing the performance of LP relaxation. In primal heuristics, AI algorithms aim to generate high-quality solutions quickly, providing tighter upper bounds for minimization problems and facilitating more pruning of the search tree. For example, Nair et al. (2021) proposed a neural diving method that used deep neural networks to generate multiple partial assignments for integer variables, reducing the problem to smaller MIPs and improving the solution process.

Recent studies have extensively employed Graph Neural Networks (GNNs) as AI algorithms to solve optimization problems, such as in Nair et al. (2021). GNNs naturally align with the permutation-invariant structure of constraint optimization formulations, making them well-suited for learning over bipartite graphs. Building on this, Chen et al. (2023a, 2023b) proved that well-designed GNNs can predict feasibility, boundedness, and approximate optimal solutions for LPs and unfoldable MILPs, with these results further extended to linearly constrained quadratic programs and their mixed-integer variants (Chen et al., 2024). Moreover, Chen et al. (2025) introduced second-order folklore GNNs (2-FGNNs) and proved that they can approximate strong-branching scores with high accuracy across general MILPs. These results provide a theoretical foundation for future research on GNN-based AI approaches for mathematical optimization.

Several challenges remain, hindering broader applications of AI methods for optimization. First, data collection and generation are major obstacles. As noted by Smith-Miles and Bowly (2015), the effectiveness of an algorithm depends significantly on the careful selection of test instances. Additionally, without strong assumptions about the data distribution, it is not easy to assess computational and sample complexity, making it challenging to collect enough diverse data to improve model performance. Second, generalization across heterogeneous instance distributions—such as those found in MIPLIB—remains problematic, with performance gains often confined to narrowly defined scenarios (Chen et al., 2023a). Third, models trained on small instances often struggle to scale, and training on large instances raises computational efficiency concerns.

5 The diverse landscape of optimization problems and solver evaluation

5.1 Vast diversity of problems

Optimization problems, particularly MIP problems, exhibit remarkable diversity. They arise in numerous industrial sectors, from logistics and manufacturing to finance and energy, where specific operational requirements and business objectives fundamentally drive the problem formulations. The process of formulating real-world problems into standard optimization problems involves an ever-expanding array of skills. This includes crucial choices regarding decision variables, objective functions, constraints, and also special structures like graphs and special ordered sets. This process itself introduces significant diversity, as different modelers might choose different formulations, apply varied mathematical skills, and accept different levels of approximations for the same underlying problem. This leads to optimization problems with vastly different characteristics and solution difficulties, even before considering the inherent complexities of solving MIPs.

Furthermore, the sheer number of variables and constraints, the distribution of nonzero elements in the problem matrices, and numerical conditioning all contribute significantly to this diversity. In summary, the interplay between external application domains, the internal mathematical structure of the problems, and human factors in modeling makes the universe of optimization problems extraordinarily versatile.

Despite commendable efforts to be comprehensive and fair, standard benchmark libraries like MIPLIB inevitably represent only a limited segment of this true diversity. Moreover, the scarcity of publicly available, real-world industrial data, often due to proprietary concerns, further constrains the representativeness of these benchmarks.

5.2 The elephant in the room: Solvers overfitting to benchmarks

Solver developers naturally use public benchmark libraries to test, refine, and showcase the performance of their software, especially given competitive pressures and marketing needs. Over time, algorithmic choices, default parameter settings, presolving routines, and heuristic strategies within the solvers evolve to perform particularly well on these known instances. This could be a critical issue when facing real-world problem instances from

industry, which are often kept confidential, limiting the variety of problems readily available.

Benchmark-driven development can indeed lead to impressive performance gains on the benchmark set itself. Indeed, developers have shown that tuning the configuration parameters of even an old-version CPLEX against the public benchmarks enabled it to the highest performance (Sun et al., 2024). However, such highly tuned performance on a given set of problems may not always generalize to new, unseen instances. Strategies that are beneficial for a broad class of problems might be turned off, or specific parameters might be set in a way that favors benchmark instances, thus leading to worse performance on problems outside the benchmark distribution.

This is particularly difficult to examine with closed-source commercial solvers, as their internal algorithms and detailed design choices are proprietary. While benchmarks are necessary and valuable for tracking progress, they might provide an incomplete reflection of solvers' performance.

5.3 End user: User-driven testing

When the selection of an optimization solver is a critical decision—impacting project timelines, solution quality, operational efficiency, and ultimately business outcomes—relying solely on published benchmark results or vendor marketing materials is insufficient. The "best" solver is context-specific, that is, performance should be measured based on the user's specific objectives, such as time to reach a sufficiently accurate solution, time to find a good feasible solution, solution quality achievable within a fixed time limit, performance robustness and reliability, required special features (e.g., infeasibility resolution, warm-starting for sequential optimization), quality of documentation, and the supplier's ability to offer software customization. A solver that is fast on average but exhibits high variability or even fails on certain instances may be less desirable than one with more reliable behavior and predictable performance.

Fortunately, the current solver ecosystem facilitates customized evaluations. Most commercial solvers offer free trial licenses or academic licenses. Standardized programming interfaces (e.g., APIs for Python, C/C++, and Java) and algebraic modeling languages allow users to model their problems once and then test them with multiple solvers.

Needless to say, it is crucial to measure performance on a set of problem instances that reflects the spectrum of sizes, structures, data distributions, and other complexities encountered in the user's application. Given that the internals of closed-source solvers are not transparent, empirical testing on the user's actual problems is the most reliable method to determine suitable selections.

Finally, solver selection is not a static, one-time decision. The field of optimization is still evolving rapidly, with continuous improvements in existing solvers and new developers entering the market.

6 Conclusions

A variety of mathematical programming solvers have been designed to address the specific optimization problems faced by industries and organizations across different sectors. Each type of problem is associated with the relevant mathematical theories, which require high levels of expertise and years of numerical software programming skills. These complexities contribute to the high development costs and barriers to entry for creating solvers. The evolution of mathematical programming solvers has been shaped not only by advancements in computer hardware but also by innovations in optimization theory.

Mathematical programming solvers were first developed by institutions and companies in Europe and the US around the 1980s, establishing a dominant position in the market. However, in the past five years, the rapid development of China-originated solvers has started to challenge the long-standing dominance of established solvers. These emerging solvers have secured leading positions in public benchmarks and gained significant recognition within the industry. While benchmarks like MIPLIB have been instrumental in driving progress, they represent only a fraction of the vast diversity of real-world problems and can inadvertently lead to solver strategies that are highly optimized for benchmarks. End-users must prioritize empirical testing on their own specific problem instances.

Recent advancements in AI and GPU technologies offer great potential for further improving solver performance. AI techniques can assist in tasks such as branching, cut selection, and primal heuristics, while GPU hardware is well-suited for parallel matrix operations, which enhance certain linear algebra computations in mathematical programming solvers. However, integrating these advancements into the current architecture of state-of-the-art solvers still faces challenges and may take time. These innovations are likely to drive a transformative shift in the overall design of mathematical programming solvers.

Competing Interests The authors declare that they have no competing interests.

References

Abhishek K, Leyffer S, Linderoth J (2010). FilMINT: An outer approximation-based solver for convex mixed-integer nonlinear programs. INFORMS Journal on Computing, 22(4): 555–567

Achterberg T, Berthold T (2009). Hybrid branching. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Berlin, Heidelberg, Springer, 309–311

Achterberg T, Bixby R E, Gu Z, Rothberg E, Weninger D (2020). Presolve reductions in mixed integer programming. INFORMS Journal on Computing, 32(2): 473–506

Achterberg T, Koch T, Martin A (2005). Branching rules revisited. Operations Research Letters, 33(1): 42–54

Achterberg T, Raack C (2010). The MCF-separator: Detecting and exploiting multi-commodity flow structures in MIPs. Mathematical Programming Computation, 2(2): 125–165

Achterberg T, Wunderling R (2013). Mixed integer programming: analyzing 12 years of progress. In: Facets of Combinatorial Optimization: Festschrift for Martin Grötschel (Jünger M, Reinelt G, Eds.). Springer, Berlin, Heidelberg, 449–481

Alibaba DAMO Academy (2025). MindOpt. Available at the website of opt.aliyun.com

Alizadeh F, Goldfarb D (2003). Second-order cone programming. Mathematical Programming, 95(1): 3–51

An L T H, Tao P D (2005). The DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems. Annals of Operations Research, 133(1–4): 23–46

Andersen E D, Andersen K D (2000). The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In: Frenk H, Roos K, Terlaky T, Zhang S, eds. High Performance Optimization. Springer US, Boston, MA, 197–232

Androulakis I P, Maranas C D, Floudas C A (1995). αBB: A global optimization method for general constrained nonconvex problems. Journal of Global Optimization, 7(4): 337–363

Applegate D, Bixby R, Chvátal V, Cook W (1995). Finding Cuts in the TSP (a Preliminary Report). DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 20: 159–173

Artelys (2024). Artelys KNITRO. Available at the website of artelys.com/solvers/knitro

Ashford R (2007). Mixed integer programming: a historical perspective with xpress-MP. Annals of Operations Research, 149(1): 5–17

Atamtürk A, Nemhauser G L, Savelsbergh M W (2000). Conflict graphs in solving integer programming problems. European Journal of Operational Research, 121(1): 40–55

Balas E (1979). Disjunctive programming. In: Annals of Discrete Mathematics Elsevier, p. 3–51

Balas E, Ceria S, Cornuéjols G (1993). A lift-and-project cutting plane algorithm for mixed 0–1 programs. Mathematical Programming, 58(1–3): 295–324

Bazaraa M S, Jarvis J J, Sherali H D (2010). Linear Programming and Network Flows. Hoboken, N.J: John Wiley & Sons

Belotti P (2009). Couenne: A user's manual. Technical report, Technical report, Lehigh University

Bengio Y, Lodi A, Prouvost A (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research, 290(2): 405–421

Benichou M, Gauthier J M, Girodet P, Hentges G, Ribiere G, Vincent O (1971). Experiments in mixed-integer linear programming. Mathematical Programming, 1(1): 76–94

Benson H Y, Sağlam Ü (2013). Mixed-Integer Second-Order Cone Programming: A Survey. In: Topaloglu H, Smith J C, Greenberg H J, eds. Theory Driven by Influential Applications. INFORMS, 13–36

Berthold T (2006). Primal heuristics for mixed integer programs. Dissertation for the Doctoral Degree. Zuse Institute Berlin (ZIB), Berlin, Germany

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

Bixby R E (2002). Solving real-world linear programs: A decade and more of progress. Operations Research, 50(1): 3–15

Bixby R E (2012). A brief history of linear and mixed-integer programming computation. In: Grötschel M, ed. Documenta Mathematica Series. EMS Press, 107–121

Bixby R E, Indovina R R (1992). MIPLIB: a test set of real-world mixed integer programming problems. Technical Report, Defense Technical Information Center, Alexandria, VA, USA. Report No. AD-A257 226

Bixby R E, Fenelon M, Gu Z, Rothberg E, Wunderling R (2004). Mixed-integer programming: a progress report. In: The Sharpest Cut: The Impact of Manfred Padberg and His Work SIAM, 309–325

Bolusani S, Besançon M, Bestuzheva K, Chmiela A, Dionísio J, Donkiewicz T, Doornmalen J van, Eifler L, Ghannam M, Gleixner A, Graczyk C, Halbig K, Hedtke I, Hoen A, Hojny C, Hulst R, Kamp D, Koch T, Kofler K, Lentz J, Manns J, Mexi G, Erik M, Pfetsch M E, Schlösser F, Serrano F, Shinano Y, Turner M, Vigerske S, Weninger D, Xu L (2024). The SCIP optimization suite 9.0. ZIB-Report, Zuse Institute Berlin (ZIB),Berlin, Germany

Bonami P, Biegler L T, Conn A R, Cornuéjols G, Grossmann I E, Laird C D, Lee J, Lodi A, Margot F, Sawaya N, Wächter A (2008). An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optimization, 5(2): 186–204

Borgwardt K H (1982a). Some distribution-independent results about the asymptotic order of the average number of pivot steps of the simplex method. Mathematics of Operations Research, 7(3): 441–462

Borgwardt K H (1982b). The average number of pivot steps required by the simplex-method is polynomial. Mathematical Methods of Operations Research, 26(1): 157–177

Boyd S (2010). Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine Learning, 3(1): 1–122

Brearley A, Mitra G, Williams H (1973). An analysis of mathematical programs prior to applying the simplex method. Mathematical Programming, 7: 263–282

Burer S, Letchford A N (2012). Non-convex mixed-integer nonlinear programming: A survey. Surveys in Operations Research and Management Science, 17(2): 97–106

Bussieck M, Vigerske S (2010). MINLP solver software. In: Cochran J J, Cox L A, Keskin Ocak P, Kharoufeh J P, Smith J C, eds. Wiley Encyclopedia of Operations Research and Management Science. Wiley, New York

Chen Z, Chen X, Liu J, Wang X, Yin W (2024). Expressive power of graph neural networks for (mixed-integer) quadratic programs. Preprint arXiv:2406.05938

Chen Z, Liu J, Chen X, Wang X, Yin W (2025). Rethinking the capacity

of graph neural networks for branching strategy. Advances in Neural Information Processing Systems, 37: 123991–124024

Chen Z, Liu J, Wang X, Lu J, Yin W (2023a). On representing linear programs by graph neural networks. arXiv:2209.12288

Chen Z, Liu J, Wang X, Lu J, Yin W (2023b). On representing mixed-integer linear programs by graph neural networks. arXiv:2210.10759

COIN-OR Foundation (2025a). COIN-OR linear programming solver. Available at the website of github.com/coin-or/Clp

COIN-OR Foundation (2025b). COIN-OR branch-and-cut solver. Available at the website of github.com/coin-or/Cbc

Conn A R, Gould N I M, Toint P L (2000). Trust Region Methods. Society for Industrial and Applied Mathematics(SIAM), Philadelphia, PA, USA

Crowder H, Johnson E L, Padberg M (1983). Solving large-scale zero-one linear programming problems. Operations Research, 31(5): 803–834

D'Ambrosio C, Lodi A (2013). Mixed integer nonlinear programming tools: an updated practical overview. Annals of Operations Research, 204(1): 301–320

Danna E, Rothberg E, Pape C L (2005). Exploring relaxation induced neighborhoods to improve MIP solutions. Mathematical Programming, 102(1): 71–90

Dantzig G, Fulkerson R, Johnson S (1954). Solution of a large-scale traveling-salesman problem. Journal of the Operations Research Society of America, 2(4): 393–410

Dantzig G B (2002). Linear programming. Operations Research, 50(1): 42–47

Dantzig G B, Fulkerson D R, Johnson S M (1959). On a linear-programming, combinatorial approach to the traveling-salesman problem. Operations Research, 7(1): 58–66

Dantzig G B, Orden A, Wolfe P (1955). The generalized simplex method for minimizing a linear form under linear inequality restraints. Pacific Journal of Mathematics, 5(2): 183–195

Dongarra J J (2022). The evolution of mathematical software. Communications of the ACM, 65(12): 66–72

Duff I S, Erisman A M, Reid J K (2017). Direct Methods for Sparse Matrices. Oxford: Oxford University Press

Duran M A, Grossmann I E (1986). An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical Programming, 36(3): 307–339

FICO (2025). Xpress. Available at the website of fico.com/en/products/fico-xpress-optimization

Fischetti M, Glover F, Lodi A (2005). The feasibility pump. Mathematical Programming, 104(1): 91–104

Fischetti M, Lodi A (2003). Local branching. Mathematical Programming, 98(1–3): 23–47

Fischetti M, Monaci M (2014). Proximity search for 0–1 mixed-integer convex programming. Journal of Heuristics, 20(6): 709–731

Floudas C A, Visweswaran V (1995). Quadratic optimization. In: Horst R, Pardalos P M, eds. Handbook of Global Optimization. Springer US, Boston, MA, 217–269

Forrest J J H, Tomlin J A (1972). Updated triangular factors of the basis to maintain sparsity in the product form simplex method. Mathematical Programming, 2(1): 263–278

Frank M, Wolfe P (1956). An algorithm for quadratic programming. Naval Research Logistics Quarterly, 3(1–2): 95–110

Friberg H A (2016). Erratum to: CBLIB 2014: a benchmark library for conic mixed-integer and continuous optimization. Mathematical Programming Computation, 8(2): 215–216

Furini F, Traversi E, Belotti P, Frangioni A, Gleixner A, Gould N, Liberti L, Lodi A, Misener R, Mittelmann H, Sahinidis N V, Vigerske S, Wiegele A (2019). QPLIB: a library of quadratic programming instances. Mathematical Programming Computation, 11(2): 237–265

Gärtner B, Matoušek J (2012). Semidefinite programming. In: Approximation Algorithms and Semidefinite Programming Springer Berlin Heidelberg, Berlin, Heidelberg, 15–25

Ge D, Huangfu Q, Wang Z, Wu J, Ye Y (2023). Cardinal Optimizer (COPT) user guide. Available at the website of guide.coap.online/copt/en-doc

Geoffrion A M (1972). Generalized benders decomposition. Journal of Optimization Theory and Applications, 10(4): 237–260

Gill P E, Murray W, Wright M H (2019). Practical Optimization. Philadelphia, PA: Society for Industrial and Applied Mathematics

Gomory R (1958). Outline of an algorithm for integer solutions to linear programs. Bulletin of the American Mathematical Society, 64(5): 275–278

Gomory R E (1960). An algorithm for the mixed integer problem. Technical report, The Rand Corporation, Santa Monica, CA, 1885

Gondzio J, Sarkissian R (2003). Parallel interior-point solver for structured linear programs. Mathematical Programming, 96(3): 561–584

Gu Z, Nemhauser G L, Savelsbergh M W P (1998). Lifted cover inequalities for 0–1 integer programs: Computation. INFORMS Journal on Computing, 10(4): 427–437

Gu Z, Nemhauser G L, Savelsbergh M W P (1999). Lifted flow cover inequalities for mixed 0–1 integer programs. Mathematical Programming, 85(3): 439–467

Gupta O K, Ravindran A (1985). Branch and bound experiments in convex nonlinear integer programming. Management Science, 31(12): 1533–1546

Gurobi Optimization (2023). Does gurobi support GPUs? Available at: Gurobi Help Center 2024–11–26 (cited that very day)

Gurobi Optimization (2025). Gurobi. Available at the website of gurobi.com

Han Q, Lin Z, Liu H, Chen C, Deng Q, Ge D, Ye Y (2024). Accelerating low-rank factorization-based semidefinite programming algorithms on GPU. arXiv:2407.15049

Hoffman K L, Padberg M (1991). Improving LP-representations of zero-one linear programs for branch-and-cut. ORSA Journal on Computing, 3(2): 121–134

Huang Y, Zhang W, Li H, Ge D, Liu H, Ye Y (2024). Restarted primal-dual hybrid conjugate gradient method for large-scale quadratic programming. arXiv:2405.16160

Huangfu Q, Hall J A J (2018). Parallelizing the dual revised simplex method. Mathematical Programming Computation, 10(1): 119–142

Huawei (2025). OptVerse.Available at the website of huaweicloud.com

Huawei Taylor Lab (2024). TAYLOR. Available at the website of tcs-lab.com/solver

IBM ILOG (2022). CPLEX optimization studio. Available at the

website of ibm.com/optimization-solver

Johnson E L, Padberg M W (1982). Degree-two inequalities, clique facets, and biperfect graphs. In: North-Holland Mathematics Studies Elsevier, 169–187

Karmarkar N (1984). A new polynomial-time algorithm for linear programming. In: Proceedings of the 16th Annual ACM Symposium on Theory of Computing. 302–311

Kelley C T (1999). Iterative Methods for Optimization. Philadelphia, Pa: Society for Industrial and Applied Mathematics(SIAM), Philadelphia, PA, USA

Khachiyan L G (1979). A polynomial algorithm in linear programming. In: Doklady Akademii Nauk. Russian Academy of Sciences 244: 1093–1096

Khuri A I, Mukhopadhyay S (2010). Response surface methodology. Wiley Interdisciplinary Reviews: Computational Statistics, 2(2): 128–149

Klee V, Minty G J (1972). How good is the simplex algorithm. In: Inequalities III Academic Press, 159–175

Klotz E, Newman A M (2013). Practical guidelines for solving difficult linear programs. Surveys in Operations Research and Management Science, 18(1–2): 1–17

Koch T, Berthold T, Pedersen J, Vanaret C (2022). Progress in mathematical programming solvers from 2001 to 2020. EURO Journal on Computational Optimization, 10: 100031

Kojima M, Mizuno S, Yoshise A (1989). A polynomial-time algorithm for a class of linear complementarity problems. Mathematical Programming, 44(1–3): 1–26

Lagarias J C, Reeds J A, Wright M H, Wright P E (1998). Convergence properties of the nelder–mead simplex method in low dimensions. SIAM Journal on Optimization, 9(1): 112–147

Lasdon L S, Waren A D (1980). Feature article—Survey of nonlinear programming applications. Operations Research, 28(5): 1029–1073

Laundy R, Perregaard M, Tavares G, Tipi H, Vazacopoulos A (2009). Solving hard mixed-integer programming problems with xpress-MP: A MIPLIB 2003 case study. INFORMS Journal on Computing, 21(2): 304–313

Lemke C E (1954). The dual method of solving the linear programming problem. Naval Research Logistics Quarterly, 1(1): 36–47

Lenovo (2024). Lenovo optimization solver (LeOPT). Available at the website of leopt.cn

Linderoth J T, Savelsbergh M W P (1999). A computational study of search strategies for mixed integer programming. INFORMS Journal on Computing, 11(2): 173–187

Lipp T, Boyd S (2016). Variations and extension of the convex–concave procedure. Optimization and Engineering, 17(2): 263–287

Lobo M S, Vandenberghe L, Boyd S, Lebret H (1998). Applications of second-order cone programming. Linear Algebra and Its Applications, 284(1–3): 193–228

Lougee-Heimer R (2003). The common optimization interface for operations research: Promoting open-source software in the operations research community. IBM Journal of Research and Development, 47(1): 57–66

Lu H, Yang J (2024a). cuPDLP.jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in Julia. arXiv:2311.12180

Lu H, Yang J (2024b). A practical and optimal first-order method for large-scale convex quadratic programming. Preprint arXiv:2311.07710

Lu H, Yang J, Hu H, Huangfu Q, Liu J, Liu T, Ye Y, Zhang C, Ge D (2024). cuPDLP-C: A strengthened implementation of cuPDLP for linear programming by C language. arXiv:2312.14832

Lustig I J, Marsten R E, Shanno D F (1994). Feature article—interior point methods for linear programming: Computational state of the art. ORSA Journal on Computing, 6(1): 1–14

Marchand H, Wolsey L A (2001). Aggregation and mixed integer rounding to solve MIPs. Operations Research, 49(3): 363–371

Markowitz H M, Manne A S (1957). On the solution of discrete programming problems. Econometrica, 25(1): 84–110

Mehrotra S (1992). On the implementation of a primal-dual interior point method. SIAM Journal on Optimization, 2(4): 575–601

Metropolis N, Rosenbluth A W, Rosenbluth M N, Teller A H, Teller E (1953). Equation of state calculations by fast computing machines. Journal of Chemical Physics, 21(6): 1087–1092

Misener R, Floudas C A (2014). ANTIGONE: algorithms for coNTinuous / integer global optimization of nonlinear equations. Journal of Global Optimization, 59(2–3): 503–526

Monteiro R D C, Adler I (1989). Interior path following primal-dual algorithms. part I: linear programming. Mathematical Programming, 44(1-3): 27–41

MOSEK ApS (2025). Mosek. Available at the website of mosek.com

Nair V, Bartunov S, Gimeno F, von Glehn I, Lichocki P, Lobov I, O' Donoghue B, Sonnerat N, Tjandraatmadja C, Wang P, Addanki R, Hapuarachchi T, Keck T, Keeling J, Kohli P, Ktena I, Li Y, Vinyals O, Zwols Y (2021). Solving mixed integer programs using neural networks. arXiv:2012.13349

Nesterov Y, Nemirovskii A (1994). Interior-Point Polynomial Algorithms in Convex Programming. Society for Industrial and Applied Mathematics. Philadelphia, PA, USA

Nocedal J, Wright S J (2006). Numerical Optimization. New York, NY: Springer

Orchard-Hays W (1990). History of the development of LP solvers. Interfaces, 20(4): 61–73

Padberg M W, Van Roy T J, Wolsey L A (1985). Valid linear inequalities for fixed charge problems. Operations Research, 33(4): 842–861

Quesada I, Grossmann I E (1992). An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. Computers & Chemical Engineering, 16(10–11): 937–947

Ralphs T, Shinano Y, Berthold T, Koch T (2018). Parallel solvers for mixed integer linear optimization. In: Hamadi Y, Sais L, eds. Handbook of Parallel Constraint Reasoning. Springer International Publishing, Cham, 283–336

Rios L M, Sahinidis N V (2013). Derivative-free optimization: a review of algorithms and comparison of software implementations. Journal of Global Optimization, 56(3): 1247–1293

Rothberg E (2007). An evolutionary algorithm for polishing mixed integer programming solutions. INFORMS Journal on Computing, 19(4): 534–541

Rothberg E (2020). How to exploit parallelism in linear and mixed-integer programming. Available at the website of

gurobi.com/events/how-to-exploit-parallelism-in-linear-and-mixed-integer-programming

Ryoo H S, Sahinidis N V (1995). Global optimization of nonconvex NLPs and MINLPs with applications in process design. Computers & Chemical Engineering, 19(5): 551–566

Savelsbergh M W P (1994). Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal on Computing, 6(4): 445–454

Schubiger M, Banjac G, Lygeros J (2020). GPU acceleration of ADMM for large-scale quadratic programming. Journal of Parallel and Distributed Computing, 144: 55–67

Sherali H D, Adams W P (2013). A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems. Springer Science & Business Media

Shinano Y, Achterberg T, Berthold T, Heinz S, Koch T, Winkler M (2016). Solving open MIP instances with ParaSCIP on supercomputers using up to 80,000 cores. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Chicago, IL, USA, IEEE, 770–779

Smale S (1983a). The problem of the average speed of the simplex method. In: Bachem A, Korte B, Grötschel M, eds. Mathematical Programming The State of the Art: Bonn 1982. Springer, Berlin, Heidelberg, 530–539

Smale S (1983b). On the average number of steps of the simplex method of linear programming. Mathematical Programming, 27(3): 241–262

Smith-Miles K, Bowly S (2015). Generating new test instances by evolving in instance space. Computers & Operations Research, 63: 102–113

SRIBD (2024). Xianpeng OPTimizer (XOPT). Available at the website of xopt.sribd.cn

Stubbs R A, Mehrotra S (1999). A branch-and-cut method for 0–1 mixed convex programming. Mathematical Programming, 86(3): 515–532

Sun M, Li T, Yin W (2024). MindOpt adapter for CPLEX benchmarking performance analysis. arXiv:2312.13527

Systems L I N D O (2025). Linear, Interactive, and Discrete Optimizer (LINDO). Available at the website of lindo.com/index.php/products/lindo-api-for-custom-optimization-application

Tang Y, Agrawal S, Faenza Y (2020). Reinforcement learning for integer programming: learning to cut. In: International Conference on Machine Learning. PMLR 9367–9376

Tao P D, An L T H (1997). Convex analysis approach to D.C. programming: Theory, algorithms and applications. Acta Mathematica Vietnamica, 22(1): 289–356

Terlaky T (2009). Twenty-five years of interior point methods. In: Decision Technologies and Applications INFORMS, 1–33

The Optimization Firm (2025). BARON. Available at the website of minlp.com/baron

Van Roy T J, Wolsey L A (1985). Valid inequalities and separation for uncapacitated fixed charge networks. Operations Research Letters, 4(3): 105–112

Van Roy T J, Wolsey L A (1987). Solving mixed integer programming problems using automatic reformulation. Operations Research, 35(1): 45–57

Vandenberghe L, Boyd S (1996). Semidefinite programming. SIAM Review, 38(1): 49–95

Vigerske S, Gleixner A (2018). SCIP: global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. Optimization Methods & Software, 33(3): 563–593

Weismantel R (1997). On the 0/1 knapsack polytope. Mathematical Programming, 77(3): 49–68

Wen B, Chen X, Pong T K (2018). A proximal difference-of-convex algorithm with extrapolation. Computational Optimization and Applications, 69(2): 297–324

Wen Z, Goldfarb D, Yin W (2010). Alternating direction augmented Lagrangian methods for semidefinite programming. Mathematical Programming Computation, 2(3–4): 203–230

Westerlund T, Pettersson F (1995). An extended cutting plane method for solving convex MINLP problems. Computers & Chemical Engineering, 19: 131–136

Wilson R B (1963). A simplicial algorithm for concave programming. Dissertation for the Doctoral Degree. Harvard University

Wolfe P (1959). The simplex method for quadratic programming. Econometrica, 27(3): 382–398

Wolsey L A (2020). Integer Programming. John Wiley & Sons, Hoboken, NJ, USA

Wright S J (1997). Primal-dual interior-point methods. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA

Yuille A L, Rangarajan A (2003). The concave-convex procedure. Neural Computation, 15(4): 915–936

Zhang J, Liu C, Li X, Zhen H L, Yuan M, Li Y, Yan J (2023a). A survey for solving mixed integer programming via machine learning. Neurocomputing, 519: 205–217

Zhang M, Yin W, Wang M, Shen Y, Xiang P, Wu Y, Zhao L, Pan J, Jiang H, Huang K (2023b). MindOpt tuner: Boost the performance of numerical software by automatic parameter tuning. arXiv:2307.08085