Last fifty years of integer linear programming: a focus on recent practical advances

François Clautiaux, Ivana Ljubić

▶ **To cite this version:**

François Clautiaux, Ivana Ljubić. Last fifty years of integer linear programming: a focus on recent practical advances. European Journal of Operational Research, In press, 10.1016/j.ejor.2024.11.018 . hal-04776866

HAL Id: hal-04776866
https://inria.hal.science/hal-04776866v1

Submitted on 12 Nov 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Last fifty years of integer linear programming: a focus on recent practical advances

François Clautiaux[1] and Ivana Ljubić[2]
1. Université de Bordeaux, Institut de Mathématiques de Bordeaux, CNRS, Inria
2. ESSEC Business School, Cergy-Pontoise, France

November 12, 2024

Abstract

Mixed-integer linear programming (MILP) has become a cornerstone of operations research. This is driven by the enhanced efficiency of modern solvers, which can today find globally optimal solutions within seconds for problems that were out of reach a decade ago. The versatility of these solvers allowed successful applications in many areas, such as transportation, logistics, supply chain management, revenue management, finance, telecommunications, and manufacturing. Despite the impressive success already obtained, many challenges remain, and MILP is still a very active field.

This article provides an overview of the most significant results achieved in advancing the MILP solution methods. Given the immense literature on this topic, we made deliberate choices to focus on computational aspects and recent practical performance improvements, emphasizing research that reports computational experiments. We organize our survey into three main parts, dedicated to branch-and-cut methods, Dantzig-Wolfe decomposition, and Benders decomposition. The paper concludes by highlighting ongoing challenges and future opportunities in MILP research.

Keywords: Combinatorial Optimization, Mixed-Integer Linear Programming, Branch-and-Cut, Dantzig-Wolfe Decomposition, Benders Decomposition

1 Introduction

Mixed-integer linear programming (MILP) is now used in the majority of operations research projects. This can be explained by the increasing efficiency of modern MILP solvers together with their improved accessibility to non-experts. MILP solvers can now solve some problems that were out of reach twenty years ago in a handful of seconds. In many practical cases, modeling is sufficient to solve the problem, and modeling through MILP models has become an essential skill taught in operations research curricula. Optimization based on integer programming models is indispensable for many business applications in transportation and logistics, supply chains, revenue management, finance, telecommunications, or manufacturing.

Integer programming is a branch of mathematical optimization in which some or all variables are restricted to be integers. If the objective function and constraints (other than the integer constraints) are given as linear functions, we talk about *integer linear programs* (ILP). If all variables are binary, we talk about 0-1 ILPs. If some variables are continuous, we talk about *mixed-integer linear programs*, MILPs.

We will start with a generic formulation of a mixed integer linear program:

$$\min\{\mathbf{c}^\top \mathbf{x} + \mathbf{d}^\top \mathbf{y} : A\mathbf{x} + B\mathbf{y} \geq \mathbf{b}, (\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^{n_x} \times \mathbb{R}^{n_y}_+\} \tag{1}$$

where A is an $m \times n_x$ matrix, B is an $m \times n_y$ matrix, $\mathbf{b}$ is a vector of size m, and $\mathbf{c}$ and $\mathbf{d}$ are vectors of size n_x and n_y, respectively. All coefficients are assumed to be rational. Continuous variables $\mathbf{y}$ are assumed to be non-negative, and integer variables $\mathbf{x}$ are assumed to be bounded. Let $P = \{A\mathbf{x}+B\mathbf{y} \geq \mathbf{b}, (\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_y}_+\}$ be the set of points satisfying the linear programming (LP) relaxation of the given MILP, and let $S = P \cap (\mathbb{Z}^{n_x} \times \mathbb{R}^{n_y}_+)$ be the set of all feasible solutions. Let $n = n_x + n_y$. We define the convex hull of S, denoted by $conv(S)$, as the set of all points that are convex combinations of finitely many points from S.

A workshop on combinatorial optimization organized in Aussois in 2008 was dedicated to the celebration of 50 years of integer programming. As a result, a book [196] has been published, reprinting some of the most influential articles published in the early years by some of the pioneers in the field. The book also included tutorials on important topics such as valid inequalities [91], decomposition methods [316], or MILP computation [235]. History of combinatorial optimization was also included in [97]. A few years later, Bixby published another article on the history of MILP computation [60]. According to [59], between 1988 and 2004, hardware got 1600

times faster, and LP solvers got 3300 times faster, allowing for a cumulative speed-up factor higher than 5×10^6, and that was already 20 years ago! Even though the speed-up factors for solving LPs have slowed down in recent years, this means that if we needed two months of running time to solve an LP in the early 1990s, we would need less than one second today. Recently, Bixby compared the machine-independent performance of two MILP solvers, CPLEX and Gurobi, between 1990 and 2020 and reported speed-ups of almost 4×10^6 [57].

We have chosen to follow up on these works. This article aims to provide an overview of the most relevant developments in the field in the last 50 years. In particular, we focus on the recent results produced after the publication of [196]. For each selected topic, we recall earlier results that are important for context and comparison, and, when possible, we refer to dedicated surveys or books for more details on classical results. For books with a broad coverage of MILP results, we recommend [24, 92, 294, 318].

Another choice we have made is to focus on the computational aspects of MILPs. Therefore, we mainly review research that aims to improve the practical performance of solvers and report computational experiments. MILP solvers rely on many components, including numerical and combinatorial algorithms. Numerical algorithms used for computing relaxations, reduced costs, and subgradients are subject to slow convergence and numerical accuracy issues. For each component focused on the combinatorial nature of the problem, such as branching, symmetry detection, heuristics, or cut selection, the goal is generally to find the right tradeoff between its computational cost and the quality of the information it produces.

Several external factors have also impacted the research in MILP in the last decades. First, companies now often use MILP solvers as heuristics, running them for a limited time. This has triggered many works dedicated to primal heuristics and also changed the branching strategies, designed not only for efficiently proving optimality but also for finding good-quality solutions early in the branching tree. Second, the dramatic improvements in artificial intelligence techniques (ML and SAT) have inspired many works. They leverage these techniques to parameterize or directly replace various oracles used during the search. Third, the availability of many faster processors and a dramatically larger memory space now allows the implementation of new strategies that would have been out of reach some years ago, such as pseudo-polynomial formulations, algorithms based on enumeration, restarts, parallel runs, or the inclusion of a significantly larger number of cuts in branch-and-cut methods.

Two sections of this survey are dedicated to important decomposition methods used in MILP: Dantzig-Wolfe and Benders decompositions. Decomposition methods have been among the most active trends of the last twenty years in integer programming. These methods exploit the structures of the MILP obtained when a reformulation technique is used, typically by enumerating the extreme points of some polyhedra. The formulations obtained typically have a better linear relaxation but a significantly larger size (i.e., often exponential or pseudo-polynomial). Practical algorithms use dynamic model generation frameworks to add only the relevant variables or constraints.

In Section 2, we describe the main ingredients used in methods based on branch-and-cut, including the algorithms featured in state-of-the-art solvers. Section 3 surveys major results related to extended formulations, focusing on Dantzig-Wolfe reformulation and column generation methods. Section 4 is dedicated to Benders decomposition. In Section 5, we draw some conclusions and list opportunities and challenges faced by the MILP community.

2 Branch-and-cut methods

In the early 50's, Dantzig, Fulkerson, and Johnson [112] introduced the subtour elimination constraints to model the traveling salesperson problem. To solve the problem in practice, they proposed to dynamically add these constraints to the model, a method that has since become known as the cutting-plane procedure. In 1958, Gomory [172] showed how to adapt Dantzig's simplex algorithm and to use cutting planes to obtain a finite algorithm for solving integer programs. In general, the cutting plane algorithms start by solving the LP-relaxation of the starting MILP model to obtain a basic feasible solution, which corresponds to a vertex of the LP-relaxation polyhedron P. If this vertex does not satisfy the integrality restrictions, the method searches for a hyperplane that separates this vertex from the set S. This hyperplane corresponds to a linear constraint (a *cutting plane*) which is added to the starting LP to exclude the vertex found. The resulting LP is resolved, and the process is repeated until some stopping criterion is met. Indeed, for general MILPs, there is no theoretical guarantee that an optimal solution can be found after a finite sequence of iterations [235]. For pure integer programs, however, Gomory [172] showed that ILP can be solved after a finite number of cutting plane iterations. In another seminal work, Land and Doig [221] proposed a branch-and-bound procedure, which is a divide-and-conquer approach that also relies on the idea of solving the LP-relaxation of the starting MILP. If the obtained solution does not satisfy integrality constraints, the search space is divided into smaller MILPs (by fixing or restricting the values of some of the integer variables). A survey on branch-and-bound methods can be found in [253]. The cutting-plane method embedded in the branch-and-bound tree, also known as *branch-and-cut* procedure (introduced by [264] in the context of TSP), is nowadays at the core of all modern mixed

integer programming solvers.

In this section, we briefly discuss major developments in the theory of valid inequalities for solving MILPs and then turn our attention to the most important ingredients of branch-and-cut algorithms related to preprocessing, probing, bounding, branching, heuristics, and reduced cost fixing.

2.1 Theory of valid inequalities

A significant portion of studies in MILPs is dedicated to valid inequalities, both from a theoretical and computational perspective. An inequality $\alpha^\top \mathbf{x} + \beta^\top \mathbf{y} \leq \gamma$ is a valid inequality if it is satisfied by any feasible point $(\mathbf{x}, \mathbf{y}) \in S$ of the MILP model given by (1). Cutting planes are inequalities that are valid for $\text{conv}(S)$ but violated by some $(\mathbf{x}, \mathbf{y}) \in P \setminus \text{conv}(S)$. Facet-defining cutting planes (namely inclusionwise maximal proper faces of $\text{conv}(S)$) are of particular interest, and in the 1990s and 2000s, we have seen a plethora of articles on facet-defining inequalities for specific polyhedra. We take the *travelling salesperson problem* (TSP) as an example. TSP is asking to find a cheapest route for a traveling salesperson who wishes to visit n given cities exactly once and then return to the home city. For the history of the TSP, see [19] and the book by [101]. Since 1954, when subtour elimination constraints were discovered [112], many important improvements of MILP-based exact methods have been achieved thanks to the development and efficient implementation of TSP-specific valid inequalities[1]. In particular, facet-defining inequalities are investigated in [263, 198, 177, 258]. Various MILP formulations are compared in [265], and branch-and-cut methods are developed in [19, 198, 264].

Besides problem-specific cutting planes, some of the cuts widely used by modern solvers exploit the nature of discrete variables together with some local structure to strengthen the starting LP relaxation. For example, the (lifted) *cover inequalities* (see, e.g., [201, 227]) are derived from viewing individual constraints in the MILP as separate knapsack problems. Other popular and effective families of cuts include *clique* [74] or *flow cover* [266] inequalities and they are nowadays standard ingredients of modern MILP solvers. More examples of cutting planes that exploit a specific problem structure (typically present in packing and covering, network design, facility location, or lot sizing problems) can be found in [244]. To generate these inequalities, the solvers heuristically try to detect cliques in conflict graphs (when, for example, among two binary variables at most one can take the value one) or more general network structures (see [7]).

A large body of research is devoted to the study of *general purpose cutting planes*, namely cuts that can be derived for an arbitrary MILP. Gomory [172, 173] laid down foundations for studying general-purpose cutting planes for MILPs, however only in the 1990s, a computational evidence that they can significantly speed up branch-and-cut-based solvers has been provided [26, 104]. We refer the reader to a tutorial on valid inequalities for MILPs by Cornuéjols [105], see also [91]. Gomory's fractional [172] and mixed-integer cuts [173] have triggered further research and the discovery of many new families of general purpose cuts, including the *intersection cuts* by Balas [23], the *disjunctive cuts* (see the recent book by Balas [24]), the *split cuts* in [102], the *mixed-integer-rounding (MIR) cuts* in [261], the *lift-and-project cuts* (see [28, 24]), or the $\{0, 1/2\}$*-cuts* [76], to mention a few.

Roughly speaking, general purpose cuts are obtained by creating linear combinations of valid inequalities and applying rounding operations (taking into consideration the integrality restrictions on variables $\mathbf{x}$). For example, for pure integer programs, the Gomory fractional [172] or Chvátal-Gomory cuts [84] take a system of valid inequalities $A\mathbf{x} \leq \mathbf{b}$ and a multiplier vector $\lambda \in \mathbb{Q}^m_+$ such that $\lambda^\top A \in \mathbb{Z}^{n_x}$, $\lambda^\top \mathbf{b} \notin \mathbb{Z}$, to derive a valid cut $\lambda^\top A\mathbf{x} \leq \lfloor \lambda^\top \mathbf{b} \rfloor$. For the $\{0, 1/2\}$-cuts, the entries of λ must be from the set $\{0, 1/2\}$. To illustrate the basic idea behind general purpose cuts, we will use *split cuts* [102] which, according to [96], are computationally the most effective for MILP solvers. Given the MILP defined by (1), and a vector $\pi \in \mathbb{Z}^{n_x}_+$ and $\pi_0 \in \mathbb{Z}$, the value of $\pi^\top \mathbf{x}$ must be integer for any $(\mathbf{x}, \mathbf{y}) \in S$. Hence, we can split the search space defined by the LP-relaxation polyhedron P into two smaller polyhedra (defining a *split disjunction*):

$$\Pi_1 = P \cap \{(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{n_x}_+ \times \mathbb{R}^{n_y} : \pi^\top \mathbf{x} \leq \pi_0\} \text{ and } \Pi_2 = P \cap \{(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{n_x}_+ \times \mathbb{R}^{n_y} : \pi^\top \mathbf{x} \geq \pi_0 + 1\},$$

where obviously $S \subseteq \overline{\text{conv}}(\Pi_1 \cup \Pi_2)$ holds ($\overline{\text{conv}}(\Pi_1 \cup \Pi_2)$ stands for the closed convex hull of $\Pi_1 \cup \Pi_2$). Split cuts can be derived as follows: if $\alpha^\top \mathbf{x} + \beta^\top \mathbf{y} - \mu(\pi^\top \mathbf{x} - \pi_0) \leq \gamma$ and $\alpha^\top \mathbf{x} + \beta^\top \mathbf{y} + \rho(\pi^\top \mathbf{x} - \pi_0 - 1) \leq \gamma$ are valid for S, where $\mu, \rho \geq 0$, then $\alpha^\top \mathbf{x} + \beta^\top \mathbf{y} \leq \gamma$ is a split cut. Conversely, all split cuts can be obtained this way [105]. The lift-and-project cuts of [25] are a special case of split cuts, in which Π_1 and Π_2 are obtained as the intersection of P with $x_i \leq 0$, respectively, $x_i \geq 1$, for some binary variable x_i. Chvátal-Gomory cuts are also a special case of split cuts, as they are obtained from a split disjunction with one empty side.

For a given family of inequalities $\mathcal{F}$ and a given point $(\mathbf{x}_0, \mathbf{y}_0) \in \mathbb{R}^n$, to determine whether a valid inequality from $\mathcal{F}$ can be used as a cutting plane, one has to solve the so-called *separation problem* [176]. The separation problem is to find an inequality in $\mathcal{F}$ that is violated by $(\mathbf{x}_0, \mathbf{y}_0)$ or to prove that no such inequality exists. For the split cuts, in the separation problem, one has to find a vector (π, π_0) so that $(\mathbf{x}_0, \mathbf{y}_0) \notin \overline{\text{conv}}(\Pi_1 \cup \Pi_2)$ and to derive a hyperplane corresponding to a split inequality, which is satisfied by any point from $\overline{\text{conv}}(\Pi_1 \cup \Pi_2)$,

[1] www.math.uwaterloo.ca/tsp/history/milestone.html

and violated by $(\mathbf{x}_0, \mathbf{y}_0)$ (or to prove that no such cut exists). The problem of finding a split set that leads to a violated cut is NP-complete, see [77]. The separation problem for Chvátal-Gomory cuts is NP-complete as well [135]. Motivated by the work of [146], where the authors model and solve the separation of Chvátal-Gomory cuts through a MILP, [29] show that the separation problem for split cuts can be solved as a parametric MILP. Both studies empirically demonstrate that a significant portion of the integrality gap for some of the difficult benchmark instances from MIPLIB can be closed when these inequalities are used.

When the split disjunction is fixed, one can generate a valid cut by solving the so-called *cut-generating LP* (CGLP). In principle, we do not have to restrict ourselves to two disjunctions, and the above-mentioned polyhedra Π_i can be defined in a more general way. Sometimes, it is possible to determine a union of polyhedra $\cup_i \Pi_i$ and its closed convex hull $\overline{\text{conv}}(\cup_i \Pi_i)$ such that $S \subseteq \overline{\text{conv}}(\cup_i \Pi_i)$. Balas' disjunctive cuts [24] are derived as valid inequalities for $\overline{\text{conv}}(\cup_i \Pi_i)$. The latter has a compact representation in a higher dimensional space which is exploited to derive efficient separation procedures for these cuts (see, e.g., [148]). Typically, given a description of polyhedra Π_i through the set of linear inequalities, the separation procedure consists of solving a CGLP using a compact formulation in a higher dimensional space [24].

Given an MILP defined by (1) and a family $\mathcal{F}$ of valid inequalities $\alpha^\top \mathbf{x} + \beta^\top \mathbf{y} \leq \gamma$ generated from the LP-relaxation P, the elementary closure $P_{\mathcal{F}}$ is a convex set obtained as the intersection of all the inequalities in $\mathcal{F}$. A comparison of some of the most relevant general-purpose cuts with respect to their elementary closures can be found in [106], see also [91]. For example, when variables are restricted to be non-negative, split cuts are equivalent to MIR-cuts and Gomory mixed-integer cuts [91]. More recent results and references related to Chvátal-Gomory closure and its generalizations can be found in [117]. The split closure can be obtained using only *intersection cuts* [16]. Different generalizations of split cuts are studied in [118], and further comparisons between (generalized) intersection cuts and lift-and-project cuts can be found in [27].

Specific implementations and computational studies involving general purpose cuts can be found in [68] (lift-and-project cuts), [153, 116] (Gomory mixed-integer cuts), [17] ($\{0, \frac{1}{2}\}$-cuts), see also [235] for more detailed important developments in the history of theory & applications of (general-purpose) cutting planes for MILPs.

As we can see, many classes of general-purpose cutting planes are available for MILP solvers. Nevertheless, the cutting plane selection and the strategy related to their separation within a branch-and-bound tree are still active areas of research (see also Section 2.4). A recent review of the different classes of available cuts, their theoretical comparison, and discussion regarding computational issues can be found in [128]. Many solvers use some kind of scoring function to rank potential candidate cuts. In the last few years, ML tools have been used to enhance this process. An exhaustive literature review on learning to select or generate cutting planes can be found in [129]. Recently, an LP-free approach for learning local cuts for ILPs has been proposed in [307].

2.2 Search strategies

Although the largest share of the research on branch-and-cut has been devoted to bounding techniques, an effective search strategy is crucial to the computational efficiency of the method. Practically speaking, most branch-and-cut methods are either based on modifying the variables' bounds or on branching on so-called special ordered sets (SOS) [34]. From now on, we focus on methods based on *trivial inequalities*, i.e., inequalities that include one variable x_i only, by creating two child nodes: one by adding constraint $x_i \leq \lfloor \bar{x}_i \rfloor$ (left branch), the other by adding $x_i \geq \lceil \bar{x}_i \rceil$ (right branch), where $\bar{x}_i$ is the value of x_i in the current fractional solution.

Search strategies, therefore, consist of two ingredients: node selection (which one among currently open nodes should be treated next) and variable selection (which variable should be selected for branching). Both procedures aim to find the right tradeoff between three conflicting objectives: keeping the number of currently open nodes reasonable, improving the value of the worst dual bound of an open node, and obtaining good-quality feasible solutions early.

Variable selection Variable selection has received much more attention than node selection. All commonly used variable selection techniques compute a *score* for each fractional variable x and choose one of highest score. This score should reflect the number of branch-and-bound nodes generated by choosing this variable for branching. This quantity cannot be computed analytically, so various proxies have been proposed in the literature. The score of a variable is computed from two intermediate scores $\ell(x)$ and $r(x)$ for the left and right branches when variable x is chosen for branching. The score of a variable is computed by aggregating these two scores, typically taking the worst score, a convex combination of the two scores, or, more often, their product:

$$\max(\epsilon, \ell(x)) \times \max(\epsilon, r(x))$$

where ϵ is a value close to 0, used to break ties when $\ell(x) = 0$ or $r(x) = 0$. In [223], the authors provide a theoretical simplified model to evaluate the size of a branch-and-bound tree, assuming one knows the gains related to each variable. They propose two scoring functions based on an analytic computation of the growth of the branch-and-bound tree related to each variable. To our knowledge, the product is still used in most solvers.

All variable selection techniques aim to find the right tradeoff between the quality of information obtained and the computational cost of acquiring it. Using local information only is rarely useful (typically, choosing the most fractional variable is not significantly more efficient than a purely random choice [6]). Most techniques are based either on a memory of the previous decisions or on various levels of look-ahead methods, which simulate the effect of branching with various levels of precision.

The two most famous methods for variable selection are *pseudocost branching* and *strong branching*. The **pseudocost** [43] of a variable is the average improvement of the objective function per unit change when this variable has been chosen for branching in previous nodes. The strength of this method is the small computing time needed to compute the pseudo-costs. Its weakness is that in early branching decisions, which are the most important, no information is available on the pseudocosts, although restart strategies mitigate this effect (see subsection on modern MILP solvers). **Strong branching** [18] has the opposite properties: it is computationally expensive but more accurate, especially in the early stages of the method. This procedure creates two branches for all variables eligible for branching and computes the lower bound related to each node. To avoid taking too long to compute, the dual simplex is generally aborted after a given number of pivots for each candidate node. A deep analysis of the performance that can be reached by strong branching is reported in [127]. The authors conclude that the performance of the method is highly dependent on the problem's structure (e.g., it is excellent for lot sizing and poor for chance-constrained portfolio optimization).

Several works build on pseudocost and/or strong branching to propose search strategies. *Reliability pseudocost branching* [6], is commonly used in MILP solvers. It mixes pseudocost and strong branching by running strong branching only for variables that are *unreliable* (either no pseudocost has been computed so far for this variable, or the number of left or right branches evaluated for this variable is too small). The efficiency of strong branching is also improved by [150], by skipping the expensive evaluation of some variables that have been detected non-useful a priori. The rationale is to avoid so-called *chimerical fractionalities*, i.e., fractional variables that can be fixed to an integer value without modifying significantly the objective function. For this purpose, when an LP is solved to compute $\ell(x_i)$ or $r(x_i)$, the LP solution obtained is also exploited to compute a first evaluation of $\ell(x_j)$ or $r(x_j)$ for another variable x_j (if its value is smaller than $\lfloor \bar{x_i} \rfloor$ or greater than $\lceil \bar{x_j} \rceil$). The authors also show that giving a larger priority to variables already chosen in previous nodes can help select a suitable subset of variables candidates for strong branching. Another path to extend strong branching is to consider the impact of the current branching decision two levels deeper than the current node [168]. This modifies significantly the tradeoff between the time needed to compute the information and the size of the branch-and-bound tree. The authors conclude that this method should not be used as a default strategy but can be helpful when hard instances are considered.

An alternative branching rule, called **hybrid branching** [3], builds on the classical pseudocost rule and four additional rules from the SAT community to compute the score of a variable. The idea is to shift the focus towards feasibility by computing scores based on the analysis of failures and by analyzing the impact of the branching choice on other variables. These scores take into account the number of variables whose domain is reduced (inference values), the history of infeasible problem creation (conflict values), the length of the conflict clauses (conflict lengths), and the number of subproblems pruned when branching on this variable.

Machine learning techniques are natural choices for variable selection: the same type of decision is made a large number of times, and the quality of a decision can be evaluated a posteriori. Early methods can be seen as precursors to works leveraging machine learning to select better branching decisions (e.g., pseudocost is already a straightforward learning technique, or in [130], the authors propose a portfolio algorithm to determine which already known branching rule should be used). In 2017, the first survey on machine learning for branching [237] cited three works for variable selection [13, 12, 206], all based on supervised learning. Since determining the size of the search tree is out of reach in general, supervised learning methods generally aim at imitating the behavior of strong branching or a combination of oracles featuring strong branching (see, e.g., [163]). In [324], the authors propose new features and learning architecture to allow a better generalization of the strategies learned. In [320], the authors extend these techniques to the case where the branching scheme considers several variables simultaneously (in which case a pure strong branching strategy would be much more expensive) for specific families of binary problems (set-packing, set-covering, binary knapsack).

Node selection Node selection is also instrumental to the efficiency of the tree search. For a precise description of the classical node selection methods, we recommend [232]. So-called static methods are based on (a combination of) depth-first search or best-first search. Solvers also alternate between diving phases, where no backtrack is performed, and traditional exploration of the branching tree. More involved methods are based on estimating the best feasible solution in a given subtree (best projection, best estimate). There is even less local information for this task, and it is not surprising that most recent methods use techniques based on learning, from basic historical rules to complex neural networks. In 2017, machine learning for node selection was surveyed in [237]. Only two papers were identified: [287] based on reinforcement learning and [180] based on imitation learning (for more details, we refer to [237]). The progress of reinforcement learning has triggered new works for node selection [321, 302]. The most recent work [218] leverages machine-learning techniques to

improve node selection. This work is inspired by [180], which was dedicated to variable selection. The authors use a reinforcement learning methodology to learn a function that compares two nodes of the branch-and-bound tree. The target scores are obtained by detecting if the node contains a known optimal solution or otherwise by running a diving procedure. To our knowledge, ML-based techniques are not yet included in state-of-the-art general-purpose MILP solvers.

2.3 Primal Heuristics

In the context of MILPs, we distinguish between *primal heuristics* (that are used within a branch-and-bound framework to improve the quality of the primal bound during the search process) and *matheuristics* (that solve MILPs or use other exact methods as subroutines to find high-quality solutions). In this article, we focus on the former ones, and we refer the reader to a recent survey on matheuristics in [70]. We provide a brief overview of *general purpose primal heuristics*, i.e., heuristics that provide feasible solutions for generic MILPs. They are an important ingredient of branch-and-bound-based methods, as better primal bounds allow the pruning of more nodes and reduce the size of the search tree. Moreover, they can be useful for presolving or reduced cost fixing (see Section 2.4) so that the quality of dual bounds or strength of the cutting planes are positively affected. According to [48], modern MILP solvers feature a double-digit quantity of primal heuristics. Most of these primal heuristics can be used as a stand-alone approach for computing feasible solutions or can be integrated within the exact solvers. The last two and a half decades have been particularly fruitful when it comes to the development of general-purpose primal heuristics, and we shortly overview some of the most promising ones.

According to their nature, we distinguish between *construction* heuristics (that construct a feasible solution from scratch) and *improvement* heuristics (that try to improve the given incumbent solution). The former ones may start with a solution of an LP-relaxation, for example. The latter ones typically explore a neighborhood of a given solution $(\mathbf{x}^*, \mathbf{y}^*)$ by changing the values of some of the coordinates. Different reference points can be used as a starting point for these heuristics: current LP-optimum, LP-optimum at the root node, or the analytic center (a point at the center of polyhedron P). Similarly, the current incumbent (best known) solution in the search tree, or another known feasible solution, is usually used as a starting point for an improvement heuristic. When choosing in which direction to conduct the search, one can consider different criteria, such as the number of violated constraints (in case the current point is infeasible), the pseudocosts (see Section 2.2), or some other statistics derived from conflict analysis. To measure the impact of primal heuristics, a new performance measure, the so-called *primal integral*, has been proposed in [4, 47] and it has been meanwhile widely adopted by the community. Primal integral takes into consideration the quality of primal solutions found during the search process and the point in time when they were found. For a recent computational study of primal heuristics inside of MILP solvers, see [49].

In terms of methodology, following the classification from [48], we highlight the following types of heuristics (and we point out that the list is not exhaustive):

- *Rounding* and *Diving* heuristics: The main idea of a rounding heuristic is to start with a fractional (LP-optimal) solution and round it to the "nearest" integer feasible solution. Too simple rounding rules will certainly result in infeasible solutions. Therefore different techniques are proposed to ensure feasibility [4]. One can also round discrete variables while taking the analytic center as a reference point [259]. For MILPs with continuous variables, once all discrete variables, say $\mathbf{x}^*$, are fixed, and a feasible solution is obtained, an LP is solved to get the best possible solution values for $\mathbf{y}$ variables. Diving (also known as relax-and-fix) heuristics mimic a depth-first-search in the branch-and-bound tree: they fix one or more variables at a time, based on some rounding criteria, and continue branching until a feasible solution is found (if any). Only a single back-tracking step is usually allowed. For a more complete list of references, see [317], where conflict analysis is used to guide the diving procedure.

- *Large neighborhood search (LNS) heuristics:* LNS is a local-search-type heuristic based on the idea of exploring a neighborhood of a given feasible solution $(\mathbf{x}^*, \mathbf{y}^*)$ by unfixing some of its variables. A famous example of a local-search heuristic is Lin-Kernighan *k-opt heuristic* for the TSP (see [182] for an efficient implementation). As the size of k increases, it becomes prohibitive to perform an exhaustive search. In this case, LNS aims at solving a simpler auxiliary MILP problem (describing the search space around a given reference point) which promises to contain high-quality solutions. When these auxiliary MILPs are solved through a nested call to a general-purpose solver, the technique is sometimes referred to as *MIPping* (see, e.g., [235]). Several approaches explore this idea in the context of MILPs. For example, *local branching* [145] starts with a feasible solution and solves an auxiliary MILP to explore the k-neighborhood of the solution at hand. If variables $\mathbf{x}$ are binary, the auxiliary MILP consists of adding the Hamming-distance constraint to the original MILP (1) with respect to the values of the given incumbent $\mathbf{x}^*$:

$$\sum_{i:x_i^*=1} (1 - x_i) + \sum_{i:x_i^*=0} x_i \leq k.$$

For general integer variables, defining the Hamming distance and the k-neighborhood requires additional auxiliary variables. In RINS [111], a common neighborhood for the current LP-relaxation solution and the incumbent solution is explored. The coinciding variable values are fixed, and the remaining auxiliary MILP is solved. An Evolutionary Algorithm from [284] involves a Crossover operator that fixes coinciding values of multiple incumbent solutions and solves auxiliary MILPs. Instead of fixing the values of integer points, the *Relaxation Enforced Neighborhood Search* for MI(N)LPs, RENS [48] takes as input a solution of continuous relaxation and solves an auxiliary MI(N)LP in which each integer variable (whose value is fractional at the current point) is restricted to take one of the two values from the nearest two integer points. Compared to other methods, RENS does not require a feasible solution to trigger the search. The *pivot-and-shift* heuristic [30] first applies rounding to a given LP solution and then explores its neighborhood by adding two constraints that limit its size. Recently, some authors also proposed to change the objective function when exploring the neighborhood. In the *proximity search* of [152], the Hamming distance is minimized subject to an explicit cutoff constraint. Alternatively, [53] propose to set the coefficients of the objective to be proportional to the solution values of the analytic center. One of the important questions in LNS is how to choose which integer variables should be fixed in the auxiliary MILP and how the search space should be decomposed. A learning approach that addresses these questions has been proposed in [301].

- *Feasibility-Pump-type heuristics:* The *feasibility pump* heuristic [141, 46] is one of the most popular primal heuristics, widely used in many applications and solvers (see [52] for a recent overview). The heuristic alternates between a rounding and a projection phase. In the former one, a fractional solution is rounded to a nearest integer (with respect to the Euclidean norm). If the obtained point is not feasible, it is then projected (with respect to the Manhattan distance) to the nearest fractional point. Thus, the algorithm constructs two sequences of points: the integer ones, which violate some of the constraints, and the fractional ones, which satisfy all the constraints but the integrality restrictions. The algorithm stops when the two sequences find an overlapping solution. *Feasibility jump* [240] is a new heuristic that resembles feasibility pump in the sense that it relaxes some of the constraints. Contrary to the above-mentioned heuristics, it does not require a reference point. This is because all but the variable bounds and the integrality constraints are relaxed and penalized in the objective function in a Lagrangian fashion. This new approach has yielded some very promising computational results.

The above list is not exhaustive, and plenty of other heuristics exist, see, e.g., [144] for a more in-depth overview. Finally, finding the "best" primal heuristic within a solver is a non-trivial task, and we observe a growing number of articles trying to address this and related questions from the ML perspective. Recent studies that focus on learning to schedule primal heuristics can be found [82, 206, 297, 303].

2.4 Modern MILP solvers

Although MILP solvers are still based on solving the linear relaxation of the problem, adding cuts, and branching, modern implementations have embedded many additional features, such as symmetry handling techniques, preprocessing, restarts, and parallel implementation.

Progress and benchmark Progresses of MILP solvers have been well documented over the years. Several analyses of the improvements of MILP solvers have been published in 2007 [58], in 2012 [60], in 2013 [9], and in 2021 [210]. The latter investigates numerically the progress made in commercial MILP solvers over a span of twenty years. The authors observed a speedup of 1000 between 2001 and 2020 (50 due to algorithms, 20 due to faster computers). The performances of many solvers are documented at `https://plato.asu.edu/ftp/milp.html`, although the three historical commercial solvers are not included anymore in the benchmark (IBM ILOG CPLEX Optimizer [194], FICO Xpress Optimization [140], and Gurobi [179]).

Non-commercial solvers are available as well. SCIP Optimization Suite, developed by Zuse Institute Berlin [295, 67] provides a generic branch-cut-and-price solver. It can handle non-linear programs and integrates constraint programming methods as well. The COIN-OR Project [156] also contains several open-source solvers for mixed-integer linear and non-linear optimization [89]. OR-Toos developed by Google [175] is another open-source software gaining more attraction from the community. Finally, HiGHs [191] has become one of the most efficient open source solvers.

Handling symmetries Many common MILP formulations involve a large number of symmetries. In this context, two solutions are symmetric if they are different in terms of variable values but represent the same solution to the original problem (i.e., some variables can be permuted without modifying the problem). This occurs, for example, when some elements are assigned to sets identified by their index (bin packing, parallel machine scheduling, vehicle routing, etc.). Symmetry is detrimental to the effectiveness of the branch-and-bound

algorithm since equivalent solutions can be explored many times in the search tree. Moreover, some branching decisions do not improve the lower bound since they just produce a solution that is isomorphic to the initial one. For many problems, ad-hoc techniques or model reformulations are used to eliminate these symmetries (see [246] for a survey of classical methods for handling symmetries in MILP, including symmetry detection methods, perturbation techniques, variable fixing, etc.).

The main classical generic algorithms dedicated to symmetry breaking in MILPs are based on the analysis of **symmetry groups** (generally the set of all possible permutations of a ground set of "equivalent" variables). The first effective methods to handle symmetry groups rely on specialized branching schemes. They consist of determining for each isomorphism class of subproblems a unique representative that will be solved. The most famous methods are known as *isomorphism pruning* [245], *orbitopal fixing* [200], and *orbital branching* [262] and its generalizations [229]. Another way to avoid symmetries is to cut symmetric solutions by adding some inequalities [229, 230, 184]. All these methods were compared computationally in [272]. This ambitious study concludes that isomorphism pruning and orbital fixing allow strong performances for special instances with a very large number of symmetries. For more classical benchmarks, the methods still bring some improvements to the results. The conclusions call for better parameter tuning methods, which would allow to turn on computational expensive symmetry handling techniques only when necessary. Symmetry-breaking inequalities are still a trendy research topic. They have been the subject of several recent works focusing on set-packing/covering/partitioning [183], subsymmetries [41]. In [231], the authors propose to enrich the conflict graph of a binary MILP by adding new conflicts based on group permutations.

Presolving, node presolving, probing, and propagation Node presolving consists of a set of algorithmic techniques used at each node of the branch-and-bound to reduce the size of the model and improve its strength by removing constraints and variables, reducing the variable domains, and modifying the constraint matrix coefficients. The term "presolving" is sometimes used for the special case of root-presolving, since it is generally heavier than the presolve at each internal node. Although they play an important role in modern MILP solvers, (node) presolving techniques are among their less documented components, and the literature on the subject has been rather scarce in the last twenty years, with the exception of a handful of works [161, 165]. A recent paper describes the techniques implemented in a commercial solver [5] and evaluates their effectiveness. Turning off all preprocessing techniques multiplies by nine the total solving time on the instances tested. Almost all preprocessing techniques lead to some improvements. Among the most useful techniques in these experiments were bounds and coefficient strengthening, implied free variable substitution, implied integer detection, reducing the number of non-zero coefficients by adding equations to other rows, clique merging, probing, and node presolve (including local bound propagation, probing, orbital branching).

For combinatorial problems where feasibility is an issue, techniques from the constraint programming / SAT world can be usefully leveraged to improve their effectiveness. MILP solvers now use some variants of constraints propagation and more involved techniques from the SAT field to improve the conflict graph [8] or generate valid inequalities based on the analysis of infeasible solutions met in the branch-and-bound tree [2].

Cutting plane selection Many different families of valid inequalities have been proposed over the years (see Section 2.1). Clearly, adding all of them is not an option since it would entail a large amount of computing time and create a huge LP formulation. Cutting plane selection raises several issues [128]: computing effectiveness measures of cutting planes, deciding when to cut, and how many cuts to add, which cuts to remove. In [95], the authors aim at finding in a pool a set of constraints that maximizes the bound improvement. They formulate this problem as a quadratic integer program, which can be reformulated as a MILP when suitable inequalities are considered. In [147], the authors ensure the diversity of the cuts by generating several optimal bases and generate cuts for each basis. Finally, it should not be surprising that the most recent works on cutting plane selection leverage machine learning techniques. A first group of authors propose to learn which cuts from a given pool should be added [268, 306, 309], by introducing a computationally expensive measure for the score of a cut and a machine learning technique to predict its value. In [228], the authors learn instead which separation algorithms should be activated at each node of the branch-and-bound tree. In [96], the authors argue that the current MILP solvers mainly generate cuts at the root node, which can be detrimental to the effectiveness of the method.

MILP restart Restart has become an important component of MILP solvers. This technique, initially used in SAT solvers, consists of aborting the tree search and running it from scratch, either with different parameters or a memory of the last run(s). The usefulness of restart strategies comes from two factors. First, there is a significant variability in a solver's performance for a given benchmark, depending on the choice of parameters, the computer, or the system architecture [236]. Second, a bad branching decision at the beginning of the tree search may lead to a huge branching tree. Therefore, it is sometimes better to restart the search from scratch and make different initial decisions if the current search tree is not promising enough. Another advantage of

restart methods is their capability to use effectively the information collected in the previous runs, such as valid inequalities, conflicts, implications, and pseudo-costs [51].

Several questions arise when one implements a restart strategy. One has to ensure diversity in the various runs, decide which information is shared between the different methods [51], and design strategies to decide when a restart should be performed, and which variant is the most promising. In [151], the authors designed a *bet-and-run* approach, in which several short runs are performed with different parameters. Then, only the run with the best score is resumed, the others are aborted. To compute this score, they proposed several indicators based on classical information from MILP solvers: lower and upper bounds, number of open nodes, lower bound improvement, and percentage of fractional variables in open nodes. Similar methods have been developed, focusing on learning the best branching decisions over several different runs [207, 149].

Parallel MILP solvers All commercial solvers have multicore or even distributed implementations. However, few papers deal with parallel MILP solvers. In [79], the authors investigate several types of information to be shared between different solvers running in parallel: lower and upper bound values, feasible solutions, branching information, and cuts. In 2016 the different parallel architectures of MILP solvers were surveyed by [279]. The authors recalled the major difficulties faced by researchers in providing efficient parallel branch-and-cut algorithms: the large amount of time spent in the root node, the difficulty in predicting the search tree, and the amount of information shared by the different nodes. Besides, almost all efficient solvers are commercial black boxes, and it is difficult for researchers to interact with the internal components of the solvers, which explains why the largest share of the published research on parallel MILP has focused on SCIP (see, e.g., [299]).

Numerically safe algorithms Many formulations suffer from ill-conditioning (i.e., large differences between matrix coefficients). This leads to numerical errors, which in turn produce infeasible or suboptimal solutions in MILP solvers. Several papers [98, 107, 134] focus on the validity of Gomory cuts, which are among the most important cut generators for MILP solvers. A research path is to produce numerically safe algorithms by using exact rational libraries [99, 100]. This comes at the cost of significantly higher computing times (20 times slower for medium instances and much more for large ones in the original experiments). However, it has to be noted that classical "inexact" solvers often benefit from incorrect bounding decisions (i.e., they cut some nodes incorrectly, drastically reducing the size of the branch-and-bound tree). This work has been substantially revised and improved in [133] by including several algorithms that are responsible for the performances of modern MIP solvers: a rational exact preprocessing phase, primal heuristics, and dual certificates from the LP solver using LP iterative refinement [170].

3 Extended reformulations and column generation

The quality of the formulation is a key factor in the efficiency of a branch-and-bound algorithm since it mainly relies on the value of the linear relaxation for pruning nodes. Classical branch-and-cut techniques improve the linear relaxation of the formulation by adding additional (possibly redundant) constraints (cutting planes).

Another way to produce stronger relaxations is to reformulate the problem by introducing a typically large number of additional variables. This section presents state-of-the-art practical methods for dealing with such *extended reformulations*.

The most popular extended reformulation is based on the *Dantzig-Wolfe decomposition*. It can be solved using a column generation algorithm based on the decomposition of the problem into two subproblems: a master problem and a column generation subproblem. It has been proposed in 1960 [114] and has been leveraged to solve a hard combinatorial problem (cutting stock) in 1961 [167]. Arguably, models that would have needed column generation had been proposed a decade before (see [310] for the story of the 0th column-generation method). Column-generation techniques have obtained state-of-the-art results for routing, parallel scheduling, and cutting and packing problems.

3.1 Basics and classical results

Before surveying more recent results on Dantzig-Wolfe decomposition and column generation methods, we introduce some notations and recall basic results required in the remainder of the section. See [126, 316] for more details on column generation.

We assume that the linear constraints can be partitioned into two subsets. To simplify the notation, we consider bounded pure integer linear programs. We consider the following initial formulation (called *compact*

formulation hereafter).

$$\min \mathbf{c}^\top \mathbf{x} \tag{2a}$$
$$A^1\mathbf{x} \geq \mathbf{b}^1 \tag{2b}$$
$$A^2\mathbf{x} \geq \mathbf{b}^2 \tag{2c}$$
$$\mathbf{x} \in \mathbb{Z}^n \tag{2d}$$

where A^1 is a $m^1 \times n$ matrix, A^2 is a $m^2 \times n$ matrix, $\mathbf{b}^1$ is vector of size m^1 and $\mathbf{b}^2$ is vector of size m^2. Let also $X^1 = \{\mathbf{x} \in \mathbb{Z}^n : A^1\mathbf{x} \geq \mathbf{b}^1\}$, $X^1_{\text{LP}} = \{\mathbf{x} \in \mathbb{R}^n : A^1\mathbf{x} \geq \mathbf{b}^1\}$, $X^2 = \{\mathbf{x} \in \mathbb{Z}^n : A^2\mathbf{x} \geq \mathbf{b}^2\}$, and $X^2_{\text{LP}} = \{\mathbf{x} \in \mathbb{R}^n : A^2\mathbf{x} \geq \mathbf{b}^2\}$. To ease the presentation, we consider that X^2 is bounded.

The Dantzig-Wolfe extended formulation is based on Minkowski's theorem, which allows for representing a polyhedron as a combination of its extreme points and rays. In our context of a bounded polyhedron, we only need extreme points. Let $\{\bar{\mathbf{x}}^p\}_{p\in P}$ be the set of extreme points of $\text{conv}(X^2)$. The so-called master problem (4) is obtained by introducing a new variable λ_p for $p \in P$. Each variable λ_p indicates the weight of point $\bar{\mathbf{x}}^p$ in the convex combination.

$$\min\{\mathbf{c}^\top \mathbf{x} : A^1\mathbf{x} \geq \mathbf{b}^1, \mathbf{x} = \sum_{p\in P} \bar{\mathbf{x}}^p\lambda_p, \sum_{p\in P} \lambda_p = 1, \lambda_p \geq 0, \forall p \in P, \mathbf{x} \in \mathbb{Z}^n\} \tag{3}$$

The master problem typically has exponentially many variables and is solved using a *branch-and-price* method. This method is a specialization of the branch-and-bound algorithm for models made intractable by their number of variables.

A *column-generation* algorithm is used to solve the linear relaxation of the master problem obtained by relaxing the integrality constraint on the original $\mathbf{x}$ variables. The original $\mathbf{x}$ variables are generally projected out by directly expressing the objective function and constraints in terms of the λ variables.

$$\min \sum_{p\in P} (\mathbf{c}^\top \bar{\mathbf{x}}^p)\lambda_p \tag{4a}$$
$$\sum_{p\in P} (A^1\bar{\mathbf{x}}^p)\lambda_p \geq \mathbf{b}^1 \tag{4b}$$
$$\sum_{p\in P} \lambda_p = 1 \tag{4c}$$
$$\lambda_p \geq 0, \quad \forall p \in P \tag{4d}$$

The algorithm alternates between solving a *restricted master problem* (RMP) using a subset of P and a so-called *pricing problem*, which computes a variable λ_p of negative reduced cost that does not belong to the RMP until no such variable exists. Let π be the vector of dual variables of constraints (4b), and θ the dual variable associated with constraint (4c), the pricing subproblem is written as follows.

$$\text{SP}(\pi, \theta) = \min\{-\theta + (\mathbf{c}^\top - \pi^\top A^1)\mathbf{x} : A^2\mathbf{x} \geq \mathbf{b}^2, \mathbf{x} \in \mathbb{Z}^n\} \tag{5}$$

In many successful applications of branch-and-price, the pricing problem can be decomposed into several subproblems. In this case, the column generation problem can be solved using one different algorithm for each subproblem. When all subproblems have the same objective and constraints, they can be aggregated into one subproblem, replacing convexity constraints by a global upper bound $K \in \mathbb{Z}_+$ on the number of solutions of these subproblems that can be used in a solution. The case with identical subproblems is generally stated as follows. We report the case where exactly K subsystem solutions have to be produced. The case where K is an upper bound is also possible.

$$\min \sum_{p\in P} (\mathbf{c}^\top \bar{\mathbf{x}}^p)\lambda_p \tag{6a}$$
$$\sum_{p\in P} (A^1\bar{\mathbf{x}}^p)\lambda_p \geq \mathbf{b}^1 \tag{6b}$$
$$\sum_{p\in P} \lambda_p = K \tag{6c}$$
$$\lambda_p \geq 0, \quad \forall p \in P \tag{6d}$$

Aggregating subproblems has several advantages: it involves solving only one subproblem per pricing phase and removes many equivalent solutions obtained by swapping solutions between two identical subproblems. However, branching is typically more difficult since the values of the original $\mathbf{x}$ variables cannot be retrieved directly from the λ variables. Below, we discuss the techniques that have been proposed to overcome this difficulty.

3.2 Column generation and Lagrangian relaxation

Dantzig-Wolfe reformulation is tightly linked with *Lagrangian relaxation*. Lagrangian relaxation consists of relaxing a set of constraints and penalizing their violation in the objective function. The relaxation is parameterized by so-called Lagrangian multipliers, which indicate the penalty incurred by the violation of the constraints. For these multipliers, we use the same symbol $\pi \in \mathbb{R}^{m_1}_+$, already used for dual variables above. For a fixed vector π of non-negative multipliers, applying Lagrangian relaxation to model (2) leads to the following problem.

$$L(\pi) = \min\{\mathbf{c}^\top \mathbf{x} + \pi^\top(\mathbf{b}^1 - A^1\mathbf{x}) : A^2\mathbf{x} \geq \mathbf{b}^2, \mathbf{x} \in \mathbb{Z}^n\} \tag{7}$$

Finding the best values for the multipliers is called the Lagrangian dual problem

$$\max_{\pi \in \mathbb{R}^{m_1}_+} L(\pi) \tag{8}$$

Classical methods for solving the Lagrangian relaxation use subgradient-based methods, such as bundle [225] or volume [33]. Solving (4) or (8) is equivalent to solving $\min\{\mathbf{c}^\top \mathbf{x} : \mathbf{x} \in \text{conv}(X^2) \cap X^1_{LP}\}$. Both families of methods have their advantages: in subgradient-based methods, there is no master to optimize, and a dual bound is obtained at each iteration. Column-generation procedures produce an upper bound for the relaxed problem at each step and are faster to converge in many cases (see [73] for a comparison of the bundle method and Dantzig-Wolfe decomposition, and [298] for a comparison of the bundle and volume method).

A disadvantage of the column generation method is that it does not produce directly a lower bound before full convergence. At each iteration, solving the master problem only produces an upper bound for the current relaxation. This can be fixed by using at each column generation step the current dual values $\bar{\pi}$ to compute the corresponding Lagrangian bound $L(\bar{\pi})$. Let $\bar{\theta}$ be the current value of dual variable θ. The value of the RMP associated with $\bar{\pi}$ is equal to $\bar{\pi}^\top \mathbf{b}^1 + \theta$. Then $L(\bar{\pi}) = \bar{\pi}^\top \mathbf{b}^1 + \theta + \text{SP}(\bar{\pi}, \bar{\theta})$, that is, the optimal value of the restricted master problem plus the value of the pricing subproblem. Although the value of the restricted master problem decreases during the column generation algorithm, this is not true for the Lagrangian bound. Recording the best dual bound obtained so far allows one to stop the method if the gap with the current primal bound is zero, even if the pricing subproblem returns a variable with a negative reduced cost.

3.3 Stabilization of column generation

From a dual point of view, the column-generation procedure is a cutting-plane algorithm, which is known to have convergence issues. Typically, one can observe large oscillations of the dual variables from one iteration to another. From a primal point of view, the restricted master problem is often highly degenerate. When the master problem is a variant of set-partitioning or a set-covering model, which is the case in many successful applications of column generation, a single column contributes to a large number of covering/packing constraints, and the number of non-zero basic variables is generally significantly smaller than the number of constraints [137].

Two main types of strategies are used to improve the convergence of column-generation algorithms. *Primal stabilization* methods focus on the selection and the number of primal columns to add to RMP, while *dual stabilization* methods aim at reducing the oscillations of the dual variable values.

Primal stabilization Primal stabilization techniques consist of adding several subproblem solutions at each iteration of the column-generation process. These solutions can be obtained heuristically by solvers producing several solutions (e.g., dynamic programming algorithms, MILP solvers, stochastic algorithms) or by running a solver with different parameters several times. To our knowledge, many implementations of column generation use primal stabilization, although few dedicated generic contributions exist. In [254], the authors compare several strategies, focusing on the notion of diversity/heterogeneity of the subproblem solutions. They conclude that producing several columns that contribute to a maximum number of constraints is beneficial for the convergence of the algorithm and significantly reduces the time needed for two classical applications: cutting stock and vehicle routing.

Dual stabilization Dual stabilization techniques aim at reducing the variations of the dual values from one iteration to the next. This can be done by forcing the dual solution to stay near the solution found at the previous iteration, either through constraints [247] or by penalizing the deviation [131]. In these methods, guessing near-optimal dual values is a clear advantage since it allows stabilizing the search in an interesting subspace. In [39], the authors empirically compare the impact of various penalty terms, concluding that a five-piece linear term is the best compromise between flexibility and implementation cost for hard instances. Dual stabilization can also be obtained by solving the pricing problem (seen as a dual separation subproblem) on a point that is not the current dual optimal solution of the restricted master problem. Such a method is called

interior point stabilization for column generation [285]. In this method, the dual point to separate is computed as the convex combination of several dual solutions obtained by applying a perturbation to the dual objective. Another variant is to use an interior-point method for solving the master problem approximately [174, 255]. In early iterations, the restricted master problem is only a rough approximation of the problem to solve, and solving it exactly is not mandatory. Moreover, interior point methods naturally produce well-centered solutions, which is a desirable behavior for stabilization.

In **in-out stabilization** (see [270]), the separation point used is a convex combination of the dual solution related to the restricted master problem and a stability center (the best feasible dual solution obtained so far). Two cases can occur: either the convex combination obtained is dual feasible, and a new best dual solution is obtained, or it is not feasible, and the separation procedure can produce a deeper cut since the separated point is closer to the dual polyhedron. An advantage of this method is that only one dual solution is used at each iteration and has few parameters. It is now used in a state-of-the-art solver for capacited vehicle routing problems [270].

Several authors proposed ad-hoc methods for stabilizing the column generation process for specific problems. These methods rely on adding **deep dual inequalities** that exclude dominated (possibly optimal) dual solutions. The most famous case is the cutting-stock problem. The dual variables of the Gilmore-Gomory model [167] have a very specific structure, which was exploited in [38] and [85]. A similar approach was used by [178] to solve a generalization of the knapsack problem. From a primal perspective, dual cuts are also presented as *exchange vectors* [315]. These vectors are special variables that allow to generate new columns implicitly by combining them with standard columns. For the cutting-stock problem, their validity comes from the fact that it is always possible to replace an item with a smaller one in a bin. In [274], the authors propose a different way to exploit the special structure of the dual of the Gilmore-Gomory model. It assumes that the function that maps the item sizes to their dual values is non-decreasing and piecewise linear. The model consists of determining the parameters of this function. The number of pieces is a parameter that is updated during the search. These techniques are generally not straightforward to adapt to new problems (the dual solutions must have very specific structures), and the improvements in the generic stabilization techniques (i.e., in-out) reduce the need for ad-hod techniques. However, recent works show that they can still be useful for other problems than variants of cutting-stock, including two-echelon vehicle routing [251], or multi-commodity network design [181] among others.

3.4 Branching

In branch-and-price methods, it is not advisable to branch directly on the master variables λ. This leads to an unbalanced tree (the right sub-tree has only one node) and harder subproblems since it is generally hard for the subproblem to avoid generating forbidden columns again.

A classical method is to branch on the values of the variables $\mathbf{x}$ of the original formulation (2). The values of these variables can be recovered by inspecting the values of variables λ: $\mathbf{x} = \sum_{p \in P} \bar{\mathbf{x}}^p \lambda_p$ (see, e.g., [313]). Let $\bar{x}_i$ be the value of a variable. If it is fractional, there are two ways to apply variable branching. The first possibility is to add directly one of the following cuts to the master problem:

$$\sum_{p \in P} \mathbf{x}_i^p \lambda_p \leq \lfloor \bar{x}_i \rfloor \quad \text{or} \quad \sum_{p \in P} \mathbf{x}_i^p \lambda_p \geq \lceil \bar{x}_i \rceil.$$

The second possibility is to remove the columns that do not satisfy the constraint and modify the subproblem to generate only solutions that satisfy them. For the left branch, the pricing obtained is as follows.

$$\text{SP}(\pi, \theta) = \min\{-\theta + (\mathbf{c}^\top - \pi^\top A^1)\mathbf{x} : A^2\mathbf{x} \geq \mathbf{b}^2, x_i \leq \lfloor \bar{x}_i \rfloor, \mathbf{x} \in \mathbb{Z}^n\}$$

Note that in the first method, the relaxed problem solved at the created node is $\min\{\mathbf{c}^\top \mathbf{x} : \mathbf{x} \in \text{conv}(X^2) \cap X^1_{LP} \cap \{x_i \leq \lfloor \bar{x}_i \rfloor\}\}$, while the second is $\min\{\mathbf{c}^\top \mathbf{x} : \mathbf{x} \in \text{conv}(X^2 \cap \{x_i \leq \lfloor \bar{x}_i \rfloor\}) \cap X^1_{LP}\}$, which is stronger. However, the second method forces the subproblem to deal with bound constraints, which is tractable in many cases but may change its complexity for some specific structures.

Branching is more complex when one considers **identical subsystems** (model 6). This is the case for many applications, where elements are assigned to subsets related to homogeneous vehicles/bins/machines. In this case, there is no direct way to retrieve the original assignment variables from the master variables (typically, the index of the vehicle/bin/machine is not indicated). There are three ways to overcome this difficulty: (1) disaggregating the subproblems, (2) finding a reformulation that allows recovering the original variable values from those of the extended formulation, or (3) branching directly on sums of the master variables.

The general case with identical aggregated subproblems has been treated by [314] using a multi-phase branching strategy. This strategy first ensures the integrality of the aggregate variables. Then, a greedy algorithm expresses the current solution using original variables. If this disaggregated solution is not integer,

aggregated convexity constraint 6c is iteratively disaggregated, forming nested partitions of the subproblem solutions. An algorithmic framework is proposed to exploit the nested structure of the subproblems.

Most successful applications of column generation with identical subproblems introduce implicit variables that allow the expression of the branching decisions using master variables only. When set-covering/partitioning problems are considered, the **Ryan and Foster** strategy [286] is generally used. This branching rule uses implicit binary variables indicating whether two elements to be covered/partitioned belong to the same set or not. Then, one branch forbids using both elements in the same subproblem solution, while the other forbids solutions where only one of the two elements is present in a solution. When the pricing subproblem can be reformulated as a (resource-constrained) shortest-path problem, it is possible to branch on the usage of some subsets of arcs in the graph (**resource branching**). In [164], the authors show that branching on subsets of arcs based on some resource consumption (i.e., time or capacity) significantly reduces the number of nodes in a branch-and-price method for a vehicle routing problem with time windows.

3.5 Primal heuristics

For difficult problems, large-scale instances are generally out of reach for branch-and-cut-and-price methods. In these cases, primal heuristics based on column generation may still be effective in producing feasible solutions. Many methods are direct adaptations of classical MILP heuristics, where an exact algorithm solves a restriction of the initial problem or a given subproblem (see Section 2 for a general discussion on MILP-based heuristics). However, using column generation methods in a heuristic has several specificities. They benefit from the fact that the column-generation procedure generates not only a primal solution but also a large number of columns, which can be recombined to produce new solutions. An advantage of primal heuristics is that they do not need to solve the pricing subproblem optimally or to reach full convergence for the column generation procedure.

The most classical method and easiest to implement is the so-called **restricted master heuristic** (see, e.g., [239]). This method consists of solving the master problem at the root node and then running a MILP solver on the last restricted master problem. Although it can produce feasible solutions in some specific cases, this method is generally not the best option.

The so-called **diving heuristic** is now considered a method of choice to produce good-quality feasible solutions [291]. These methods are based on a truncated tree search. The branching scheme used in diving methods is generally not the same as the one used in a complete search. A usual choice is to fix the value of one column (generally based on its value in the optimal solution of the linear relaxation). This branching scheme does not impact the pricing subproblem.

If the time spent solving the master problem is the largest share of the computing time, then alternative methods may be advisable. Subgradient-based approaches do not require solving a master problem and can be used to find good dual solutions. In this case, recovering a complete solution is difficult since the columns/subproblem solutions obtained at each iteration are all generated using the same set of multipliers and are likely to cover only a subset of the master constraints. A typical solution is to record solutions obtained in the previous iterations and either solve a restricted master problem using only a subset of these columns or use a greedy heuristic. For a survey on Lagrangian heuristics, we refer the reader to [71].

3.6 Modern branch-(and-cut-)and-price solvers

To go from a textbook branch-and-price to modern branch-and-cut-and-price (BCP) solvers, one needs to design stabilization strategies, branching strategies, efficient column generation, variable fixing, primal heuristics, and cut generation that can be handled by the pricing subproblem. For a recent overview of modern branch-and-price implementations, we recommend [288].

Branch-and-cut-and-price Due to the typically long time spent at each node, BCP methods are mostly useful when the integrality gap is small. When the reformulation is not enough to obtain a good enough LP relaxation, cuts are added to the master problem. However, each cut adds an additional dual variable and may impact the pricing subproblem (typically, they induce additional costs related to the values of subsets of variables). A traditional practice is to avoid cuts that have a significant impact on the pricing subproblem. Cuts that do not modify the difficulty of the pricing are called *robust* (see, e.g., [157]). However, this approach has evolved in the most recent works. Nowadays, it is common to use cuts that break the structure of the subproblem as long as efficient algorithms are able to cope with its increased complexity. This is typically the case for label-setting algorithms. This has inspired, for example, the limited-memory rank one cuts [269], which are hand-tailored for limiting the impact on dynamic-programming-based algorithms. Although each non-robust cut produces an additional resource constraint and, thus, weakens the dominance relations between labels, they can produce state-of-the-art results for various classes of vehicle routing problems [289].

Pricing strategies Pricing strategies are also key to the efficiency of BCP algorithms. Solving the pricing subproblem with a commercial MILP solver is unlikely to yield competitive results, even compared to a straightforward compact formulation. In many cases, pricing algorithms are hand-tailored for a specific application, such as vehicle routing [289] or scheduling [300]. However most efficient methods share some common ingredients. First, using an exact pricing algorithm is rarely useful, especially in the first iterations where the dual variables are far from their optimal values, and the first columns are not likely to be part of the final LP solution. Using fast heuristic procedures is generally a good choice in the first iterations [162, 123]. When the goal is to obtain a dual bound to cut a node, solving a relaxation of the subproblem is also possible. This is recommended when finding a globally optimal solution is computationally prohibitive, and a sufficiently good approximation can be obtained by relaxing some hard constraints. In some cases, it is possible to obtain a model by adding these constraints to the master problem. This is the case for vehicle routing problems with *elementarity* constraints, which ensure that each customer is visited exactly (at most) once (see [32]). Finally, similar to branch-and-bound, branch-and-price methods suffer from numerical errors. The pricing subproblem may slightly overestimate the reduced cost of some variables, leading to a non-valid dual bound. Techniques for computing safe dual bounds are discussed in [158].

Strong branching Strong branching is a key ingredient in branch-and-cut solvers [269]. Many methods try to avoid using strong branching too often since it entails a large amount of computing time. For BCP methods, the tradeoff is different. On the one hand, evaluating a branching decision is much more expensive since the pricing subproblem has to be called repeatedly. On the other hand, more time is spent at each node, and the branching tree is expected to be smaller, so any "bad" branching decision has a strong impact on the computing time. Modern BCP solvers use a *hierarchical strong branching* procedure [269]. A first rough evaluation is performed on each variable candidate for branching (no additional column generation is performed). Then, a second phase uses a heuristic pricing algorithm to obtain a more precise evaluation of the best candidates. A memory of the best branching decision is also kept (similar to pseudo-costs).

Master variables enumeration Compared to the classical branch-and-cut method used in MILP, most preprocessing, automatic cut generation, and advanced branching strategies are not available in branch-and-cut-and-price since many techniques rely on knowing the full constraint matrix. Therefore, if at some point in the algorithm, it is possible to generate *all* useful remaining columns in the master problem and solve directly the master problem using a MILP solver, it is generally beneficial for the method. In [31], the authors proposed to generate all columns with a reduced cost that is smaller than the current gap. Then, the master problem can be solved directly using a general-purpose MILP solver instead of relying on an iterative column generation process. This technique can be used when the number of columns satisfying the gap condition is not too large, and one can generate all of them efficiently (typically using a dynamic programming method that computes all non-dominated solutions in one pass). This technique is used in state-of-the-art VRP solvers based on column generation.

Specialization of reduced-cost fixing to branch-and-price Most modern branch-and-cut-and-price techniques make use of specialized *reduced cost fixing* procedures. They can be applied to subproblem variables or directly to master columns when master variables enumeration is used. To fix the value of some subproblem variables, one needs to conceive a probing method based on an oracle to compute a dual bound for the objective value when the variable takes a given value. Fixing subproblem variable values allows for faster pricing problems and fewer potential variables with a fractional value. The effectiveness of these methods relies on the possibility of determining the reduced cost of each variable in one pass (e.g., by using forward-backward dynamic programming techniques [271]).

Similarly to what is done when Lagrangian multipliers are used, using optimal dual values is not always the best choice to remove the largest number of variables. In [319], the authors study so-called *deluxing methods*, which aims at improving these algorithms. The best filtering would be obtained by choosing the best dual values for each individual variable. This would be computationally prohibitive. The other extreme is to maximize the sum of the reduced costs by selecting a dual solution in the optimal face. The authors propose a strategy that consists of clustering the master variables and computing a suitable dual value for each cluster. It allows the elimination of a large number of variables for several variants of vehicle routing problems.

3.7 Extensions of the Dantzig-Wolfe reformulation

The textbook Dantzig-Wolfe decomposition relies on the Minkowski theorem and uses a convex combination of extreme points of a polyhedron. Several researchers have studied the possibility of extending the column generation algorithm, either by using alternative reformulation techniques [290, 256], or a different algorithmic framework for solving the Dantzig-Wolfe reformulation [137]. Other notable extensions aim at applying

such decomposition when the system does not admit a natural block decomposition, using a column-and-row generation algorithm [257].

Using linear combinations of extreme points A possible extension of Dantzig-Wolfe reformulation is to use a linear combination instead of a convex combination of extreme points. This has been proposed for the special case of network-flow formulations under the name *column generation for extended formulations* [290]. In this work, the authors address problems that admit a network-flow formulation with additional constraints where the number of arcs is prohibitively large. The basic idea is to generate the arcs dynamically by generating a path by solving a classical pricing algorithm and splitting the path into individual arcs that are added to the restricted master program. The flow-conservation constraints corresponding to the new arcs are added to the restricted master problem using a column-and-row generation algorithm. A similar idea has been developed in [55] and is known as the *Bienstock-Zuckerberg (BZ) algorithm*. This method has been cast as a variant of the column-generation method in [256]. The idea is similar to [290]. Instead of ensuring that the solution belongs to $\text{conv}(X^2)$, the additional constraints ensure that the solution belongs to the linear space spanned by solutions of X^2. Practically speaking, this means that the solution can be obtained by a linear combination of some points forming a basis for X^2. The method begins with a subset of these points, and at each iteration, a linearly independent point is added until the current basis allows to generate an optimal solution. The techniques used in [290] and [256] produce formulations whose linear relaxations are weaker than the one obtained by the Dantzig-Wolfe formulation and with a larger master problem since they include the original constraints. The first successes of this method were obtained on problems where the constraint matrix A^2 of the subproblem is totally unimodular. In these cases, the linear relaxation is the same as the one obtained using the Dantzig-Wolfe reformulation. The strength of the method is the significant reduction in the number of iterations compared to standard column generation. This reduction can be explained by the fact that a large number of subproblem solutions are implicitly generated by combining smaller pieces (typically, a set of arcs of polynomial size can produce an exponentially large number of paths).

Dynamic Constraint Aggregation (DCA), Improved Primal Simplex (IPS) Another notable line of work aims at improving BCP by proposing a different algorithmic framework for column generation [137, 136]. The aim is to reduce the number of iterations needed to converge, but the modifications to the column-generation framework are too significant to consider them as stabilization techniques. The most famous variant is called *Improved Primal Simplex (IPS)* [136], inspired by an early version of the algorithm called *Dynamic Constraint Aggregation* [137]. This method exploits the fact that the restricted master bases are generally highly degenerate. The idea is to create a *reduced master problem* involving fewer variables and constraints by discarding columns corresponding with degenerate variables and then rows that become redundant when these variables are removed. This technique is of particular interest for set-partitioning problems where the sets correspond to special structures (i.e., planning, routing), and some clusters of elements tend to be always selected together. The modified column generation algorithm first seeks a set of variables with negative reduced costs that are compatible with the current reduced problem (i.e., a convex combination of these variables does not violate the removed constraints). If no such column exists, the reduced problem is updated, and the algorithm is repeated. Many techniques have been proposed over the years to improve the first versions of the methods (see, among others, [125, 322, 124]).

Column-and-row generation In [257], the authors propose a generalization of the column-generation procedure to address problems that do not admit a natural block decomposition. When designing a column-generation procedure, one generally tries to reduce the number of constraints in the master problem as much as possible. When many constraints are needed, a common approach is to use a branch-and-cut-and-price approach, where rows are only added when needed to ensure the feasibility of the current master solution. As explained above, more rows mean more dual variables for the master and potentially a harder subproblem. In the case of knapsack constraints, where all coefficients are non-negative, if one relaxes some constraints, the reduced cost of the columns can only be underestimated, and thus priced-out columns may be non-useful after the cuts are generated. This may cause additional iterations, but it allows for keeping a tractable subproblem and is generally the most efficient way to proceed. Column-and-row generation is needed when the problem's decomposition leads to two non-independent subsystems (i.e., there exist some linking constraints between them). In this case, the reduced cost of the columns may be overestimated, and thus, the column generation procedure may converge to a non-optimal solution. In [257], the authors propose an algorithm for this case where the pricing problem is decomposed into three subproblems: one for each of the two subsystems (similar to Dantzig-Wolfe) and a third one that generates variables from both subproblems with their linking constraints. This problem entails a significantly harder problem since the third pricing subproblem is a column-and-row generation.

Dual heuristics Dual heuristics aim at computing a dual solution for the master problem without using column generation. They can warm-start stabilization methods that rely on a feasible dual solution and can also compute fast dual bounds. One of the first dual heuristics has been proposed in [139] for solving cutting-stock problems by proposing functions that map the item sizes to values in $[0, 1]$ that form a feasible dual solution for the formulation of [167] for cutting-stock. The concept was popularized under the name *dual-feasible functions* and surveyed in [14], and the most recent results are proposed in [212]. All these results, dedicated to variants of bin-packing problems, are difficult to generalize to other problems (they necessitate a formulation that is strong and with a simple structure that can be exploited through greedy algorithms). To our knowledge, very few papers proposed a generalization of these techniques to different classes of problems (see, e.g., [275]). Not surprisingly, machine learning techniques were recently leveraged for computing dual solutions. [214]. This seems to be a promising path for generalizations to other problems that were hard to reach through dedicated heuristics (provided structures or optimal dual solutions are the same in small and large instances).

3.8 Network-flow formulations based on dynamic programs

We now focus on extended formulations obtained by replacing some constraints with their *dynamic-programming reformulation* (see [90]). These formulations involve typically exponentially or pseudo-polynomially large network-flow subsystems. A classical way to solve them is to use the Dantzig-Wolfe decomposition (path-flow formulations), where the subproblem is a resource-constrained shortest-path problem. An alternative method formulates directly the problem using arc variables (arc-flow formulations). In this section, we give a broad overview of these formulations, focusing on modeling paradigms, main algorithms, and the extension to hypergraphs. More details can be found in a dedicated survey [119].

Building network-flow formulations An asset of network flow formulation is the possibility to express them using a simple formalism based on recursive formulations. They allow for proposing modeling frameworks that can express large classes of problems (typically routing, scheduling, or packing problems) and provide a set of algorithmic tools. The first task when modeling with network flows is to express some subproblems as a shortest-path problem in a graph. Then each shortest path subproblem is expressed as a linear program where each variable represents the flow through an arc. For example, in the cutting-stock problem, an arc corresponds with packing an item at a given position. A path from the source to the sink of the graph represents a sequence of items from position 0 to the end of the bin [311]. For the vehicle routing problem, the arcs correspond with going from one customer to another [271]. Therefore, a path is related to a route, and a flow is a set of routes for the vehicles. Below we survey three different formal methods that are used to produce such graphs.

First of all, **dynamic programming** is a classical way to produce stronger reformulations (see, e.g., [90]). The basic idea is to identify in a MILP a subset of variables and constraints that can be reformulated as a dynamic program (DP). This DP is cast as a shortest-path problem in a graph, where vertices are states of the dynamic program, and arcs are transitions. The extended formulation is obtained by associating a variable to each arc and creating a set of flow conservation constraints.

The notion of extended formulations based on recursive formulations has also been studied through the lenses of so-called **Decision Diagrams**. Decision diagrams [44] are labeled directed acyclic graphs (U, A, d), where U is the node set, A the arc set, and d are arc labels. The set of nodes U is decomposed into layers $L_1, \ldots, L_{n+1}$. Each layer L_i is associated with a variable x_i of an original optimization problem. Each arc whose origin is in layer L_i is labeled by a possible value for variable x_i. Therefore, each path from the source to the sink corresponds with a variable assignment that can be recovered by collecting the labels along the path. Since decision diagrams have a graph representation, similar to dynamic programs, they can be used to generate MILP models with the same properties [216, 293].

Another way to specify an arc-flow formulation is to express some subsystems using **regular languages** (see, e.g., [109]). The language is described by building a deterministic finite automaton, where the symbols/letters correspond with assigning a value to a variable, and the states are similar to those used in dynamic programming. Finding a sequence of letters recognized in the automaton can be cast as seeking a path from the source to the sink node in a directed acyclic graph. For finite languages, the graph problem produced is equivalent to the one obtained using dynamic programming.

Arc-flow vs. path-flow formulations We call a formulation an **arc-flow formulation** when arc variables (and the corresponding flow conservation constraints) are explicitly expressed in the model. To our knowledge, the first practical successful implementation of such an approach is described in [311] for the cutting-stock problem. These formulations allow fully exploiting the preprocessing methods, heuristics, and efficient branching strategies embedded in MILP solvers, which are more difficult to obtain when branch-and-cut-and-price algorithms are used. Their main limitation is the size of the graph needed to embed all possible sequences of decisions. When resource constraints are involved, these models have exponentially many variables and con-

straints. Since [311], several authors have proposed arc-flow models for scheduling [213], routing [86] or variant of packing problems [72].

Path-flow formulations are obtained from arc-flow formulations by applying a Dantzig-Wolfe formulation, where the network-flow constraints are sent to the subproblem, and the remaining constraints are kept in the master. When the paths in the network-flow formulation are subject to resource constraints, path-flow formulations generally become more powerful since efficient labeling algorithms can be designed to handle the resource constraints in a column-generation process. These formulations have an exponential number of variables but a polynomial number of constraints (typically set-partitioning/covering/packing constraints in vehicle routing [271], scheduling [300], or cutting-stock [167].

The arc-flow and path-flow formulations of the same problem have the same LP relaxation since the constraint matrices of the network-flow subsystems have the totally unimodular (TU) property. Path-flow formulations are generally preferred when the underlying graph is of exponential size. However, exponentially large networks may lead to competitive results when the combinatorial explosion is controlled [249]. Recent experiments show that column-generation methods tend to be more efficient at computing the initial linear relaxation [120]. On the opposite, branching is generally less efficient in column-generation-based methods. Hybrid methods [120] make use of both types of formulations (arc-flow and path-flow) to devise efficient algorithms.

Node aggregation of network-flow formulations When the size of the network is too large, several relaxation techniques can be leveraged to reduce its size, such as the *state-space relaxation* or the *surrogate relaxation*. The notion of **state-space relaxation** comes from the field of dynamic programming. The concept itself emerged in the 80s [83], and the possibility to modify successively the state space to improve the current dual bound has been proposed for the first time in [193], under the name *Successive Sublimation Dynamic Programmming* (SSDP), using shortest path algorithms in extended graphs, and later as *Decremental States Space Relaxation* (DSSR) [283] in labeling algorithms. State-of-the-art methods for vehicle routing problems use the latter. To refine the state space of a dynamic program, the current solution (a path in a graph) is analyzed, and if it violates some constraint, the state space is updated to forbid its violation. The same technique can be used for network-flow formulations. In this case, the technique is similar to applying surrogate relaxation to the flow conservation constraint in the graph. To our knowledge, the only successful uses of such technique to solve MILP problem are based on **discretization** [86, 66]. The most popular implementation of this idea has been coined *Dynamic Discretization Discovery (DDD)* [66]. These techniques address problems formulated as network-flow problems with a pseudo-polynomial number of nodes corresponding with the value of some resource (time, knapsack, length, etc.). They first build an aggregated network where each node represents several possible resource values. Then the problem is solved on this small network. If the solution obtained can be disaggregated into a feasible one of the same cost, the initial problem is solved. Otherwise, the network is disaggregated, and the method is rerun until the gap is closed.

Extension to hypergraph-based dynamic programs Although the extension of network-flow formulation to so-called decision hypergraphs has been known for decades [248], few works have used this possibility: [87, 138] (explicitly) and [121] (implicitly). Graph-based dynamic programming model subsystems that can be expressed as regular languages, while decision hypergraphs allow modeling algebraic languages. Algebraic grammars are a powerful modeling tool that has already been exploited in column-generation methods [109]. This capability comes with an additional cost. While constraint matrices of network-flow models in graphs have the TU property, this is no longer the case for network-flow models in hypergraphs. These models have the totally-dual integral (TDI) property, which means that solving their LP relaxation is sufficient to solve the integer version, but adding bounds on the variables breaks this structure. The rarity of hypergraph-based approaches seems to indicate that regular languages are sufficient to model many practical problems.

4 Benders decomposition

Since its introduction by Jaques Benders in the 1960s [40], Benders Decomposition has proven to be useful for a large number of MILP applications. This applies to various facility location problems [143, 142, 103, 94, 132, 159], network design problems [108], or scheduling [188, 186]. Besides, in two-stage stochastic optimization, the method is known as the *L-shaped method* [312, 56], and it belongs to state-of-the-art methods for problems in which the second-stage problem can be stated as an LP [62, 233, 224]. The method decomposes the original MILP into a master problem and one or multiple subproblems, leveraging the fact that the problem may become much easier to solve once the master variables are fixed. The method has been originally proposed for problems in which an LP needs to be solved once the discrete variables (which remain in the master) are fixed. Benders cuts are derived using the LP-duality theory. The concept has been later generalized by Geoffrion [166] to deal with convex continuous subproblems. The *generalized Benders cuts* of Geoffrion are derived using the Lagrangian duality theory. More recently, logic-based Benders decomposition has been introduced by Hooker

(see the recent book [186] for further details). This method allows one to tackle subproblems of a more general nature (e.g., discrete, non-linear problems) and relies on logical inference instead of duality theory for convex optimization.

4.1 Basic idea

Benders decomposition relies on a dual idea with respect to Dantzig-Wolfe decomposition. We are given a MILP model (1) with two sets of variables $\mathbf{x}$ and $\mathbf{y}$, that we will restate as follows:

$$\begin{aligned} \min\, & \mathbf{c}^\top \mathbf{x} + \mathbf{d}^\top \mathbf{y} \\ \text{s.t. } & A\mathbf{x} + B\mathbf{y} \geq \mathbf{b} \\ & \mathbf{x} \in X \\ & \mathbf{y} \in \mathbb{R}^{n_y}_+ \end{aligned} \tag{9}$$

Variables $\mathbf{x}$ are discrete and bounded, X is a finite set obtained as an intersection of a polyhedron with $\mathbb{Z}^{n_x}$. Once variables $\mathbf{x}$ are fixed, we obtain a linear programming problem in $\mathbf{y}$, which may even decompose into smaller subproblems for subsets of $\mathbf{y}$. Since X is a finite, discrete set, the problem can be expressed as

$$\min_{\bar{\mathbf{x}} \in X} \{\mathbf{c}^\top \mathbf{x} + \min_{\mathbf{y} \geq \mathbf{0}} \{\mathbf{d}^\top \mathbf{y} : B\mathbf{y} \geq \mathbf{b} - A\bar{\mathbf{x}}\}\}.$$

For a given $\bar{\mathbf{x}} \in X$, let $\Phi(\bar{\mathbf{x}})$ be the value function of the so-called *Benders subproblem*:

$$(P) \quad \Phi(\bar{\mathbf{x}}) = \min_{\mathbf{y} \geq \mathbf{0}} \{\mathbf{d}^\top \mathbf{y} : B\mathbf{y} \geq \mathbf{b} - A\bar{\mathbf{x}}\} \tag{10}$$

Let us look at the dual of this problem:

$$(D) \quad \Phi(\bar{\mathbf{x}}) = \max_{\mathbf{u} \geq \mathbf{0}} \{(\mathbf{b} - A\bar{\mathbf{x}})^\top \mathbf{u} : B^\top \mathbf{u} \leq \mathbf{d}\}.$$

The dual polyhedron

$$F = \{\mathbf{u} \in \mathbb{R}^m : B^\top \mathbf{u} \leq \mathbf{d}, \mathbf{u} \geq \mathbf{0}\}$$

does not depend on $\bar{\mathbf{x}}$. By Farkas' lemma [92], either the problem (P) is feasible, or the system $B^\top \mathbf{u} \leq \mathbf{0}, \mathbf{u} \geq \mathbf{0}$ has a solution with $(\mathbf{b} - A\bar{\mathbf{x}})^\top \mathbf{u} > \mathbf{0}$. In the latter case, there exists an extreme ray $\mathbf{u}^r$ such that

$$(\mathbf{b} - A\bar{\mathbf{x}})^\top \mathbf{u}^r > \mathbf{0} \text{ and hence } \Phi_D(\bar{\mathbf{x}}) \to \infty.$$

Thus, to prevent points $\mathbf{x} \in X$ that render the Benders subproblem infeasible, we need to ensure that:

$$(\mathbf{b} - A\mathbf{x})^\top \mathbf{u}^r \leq \mathbf{0},$$

for all extreme rays $\mathbf{u}^r$, $r = 1, \ldots, R$ of F. Otherwise, if the problem (P) is bounded and feasible (i.e., the given $\bar{\mathbf{x}} \in X$ is a feasible decision) the optimal value is attained at one of the extreme points of the polyhedron F. Let $\mathbf{u}^p$, $p = 1, \ldots, P$ be the set of these extreme points, then:

$$\Phi(\bar{\mathbf{x}}) = \max_{p=1,\ldots,P} \{(\mathbf{b} - A\bar{\mathbf{x}})^\top \mathbf{u}^p\}. \tag{11}$$

Putting this together, and after introducing an auxiliary variable θ to replace the objective function value associated to the term $\mathbf{d}^\top \mathbf{y}$, we obtain a reformulation of the original problem:

$$\min\, \mathbf{c}^\top \mathbf{x} + \theta \tag{12a}$$

$$\text{s.t. } \mathbf{x} \in X \tag{12b}$$

$$\theta \geq (\mathbf{b} - A\mathbf{x})^\top \mathbf{u}^p, \quad p = 1, \ldots, P \tag{12c}$$

$$\mathbf{0} \geq (\mathbf{b} - A\mathbf{x})^\top \mathbf{u}^r, \quad r = 1, \ldots, R \tag{12d}$$

Constraints (12c) and (12d) are called *Benders optimality cuts* and *Benders feasibility cuts*, respectively. Thus, we have reformulated the problem (1) in terms of $\mathbf{x}$ variables only (with an addition of a single auxiliary variable θ), and the continuous variables $\mathbf{y}$ are *projected out*. The number of variables is reduced at the expense of getting a model with a potentially exponential number of constraints. These constraints are dynamically separated in a cutting-plane fashion. This means that the master problem is initialized with a small subset of Benders cuts (which could be possibly empty). The optimal solution of the master $(\mathbf{x}^*, \theta^*)$ is then used to solve the Benders subproblem and get the value of $\Phi(\mathbf{x}^*)$. If the resulting subproblem is infeasible, an extreme ray associated with $\mathbf{x}^*$ generates a cut of type (12d), which is added to the master problem. If, otherwise

$\theta^* < \Phi(\mathbf{x}^*)$, an extreme point at which the optimal value of $\Phi(\mathbf{x}^*)$ is attained generates a new cut of type (12c) which is added to the master problem. Finally, if $\theta^* \geq \Phi(\mathbf{x}^*)$, the algorithm terminates with an optimal solution. This original approach was a multi-tree approach in which the master problem is resolved as a MILP each time a new cutting plane is generated. It is not difficult to see that at each iteration, the set of extreme points or the set of extreme rays for which Benders cuts are generated is augmented with one new element. Hence, after a finite number of iterations, the algorithm terminates.

During the iterative process, we maintain the lower-bound sequence

$$LB = \mathbf{c}^\top \mathbf{x}^* + \theta^*$$

which is non-decreasing (as more and more restrictions are added to the master problem). At the same time, in each iteration, we can also calculate an upper bound by simply evaluating the value for the subproblem for a given choice of x^* as

$$UB^* = \mathbf{c}^\top \mathbf{x}^* + \Phi(\mathbf{x}^*).$$

The upper bound sequence may have a zig-zag behavior, and therefore we have to keep track of the best feasible solution found so far by setting $UB = \min\{UB, \mathbf{c}^\top \mathbf{x}^* + \Phi(\mathbf{x}^*)\}$.

How successfully Benders decomposition can be applied to a given problem depends on many factors. According to our empirical experience, we wish to highlight the following important elements:

- As observed by many authors (see, e.g., [241]), **the quality of LP-relaxation** of the starting compact model (1) plays an important role. This is because the LP-relaxation value of the Benders reformulation (12) remains the same as the LP-relaxation of the starting MILP (1) (provided that all violated Benders cuts are separated). Hence, in case of Branch-and-Benders cut implementation (see below), the size of the branching tree will be directly affected by the choice of the starting model from which we wish to project out $\mathbf{y}$ variables. In many network design or routing applications, variables $\mathbf{y}$ represent some kind of a network flow with two, three, or even more indices. Typically, these extended (but compact) formulations provide tighter lower bounds (see [273] for Steiner trees or [21] for the inventory routing, for example). However, due to the large number of $\mathbf{y}$ variables, these tight models may still be intractable for most of the black-box solvers. Using Benders decomposition instead, one would project out these variables and "thin-out" the starting model. Nevertheless, the size of the Benders subproblem (and thus the execution time for separating Benders cuts) will be affected by the way how this flow is modeled. Hence, there is a trade-off between the choice of a tight extended MILP formulation (1) and the computational burden in solving Benders subproblems and the performance of the overall decomposition procedure.

- The **relative size of vector y compared to that of x** is important as well. The larger the number of variables that will be projected out, the higher the chances of beating the black-box solver applied on the compact formulation (1). Indeed, this is particularly true for stochastic optimization problems, where the number of second-stage variables increases with the number of scenarios.

- For problems in which the matrix B has a block diagonal structure, Benders subproblem can be further decomposed into smaller problems. We say in that case that Benders subproblem is *separable*. This frequently happens in facility location problems [142, 234], network design [108], or two-stage stochastic optimization [110, 280]. **Separability of Benders subproblem** may also speed up the solution process, as many smaller LPs need to be solved (possibly in parallel) instead of a single large one. Some recent success has been reported for facility location problems [142] where the separability property is not given, but in principle, for unseparable Benders subproblems, more problem-tailored techniques need to be applied to increase the chances of beating a black-box solver. It can be sometimes advantageous to keep some of the continuous variables in the master problem in order to achieve separability of the Benders subproblem.

- Finally, the availability of **closed formulas** or **problem-specific combinatorial algorithms** that can solve Benders subproblem much faster than using LP formulations can also bring a significant advantage over a vanilla implementation. This has been confirmed in numerous applications, and we mention some recent results for facility location [143, 234, 103] and routing [11].

4.2 Branch-and-Benders Cut

With the advance of general-purpose MILP solvers in the recent two decades, it became more efficient for many applications to consider a single-tree implementation of Benders decomposition (also known as the Branch-and-Benders-cut approach). The method starts with LP-relaxation of the Benders reformulation (12) (with a small or empty subset of Benders cuts) and separates Benders cuts at the root node of the branching tree. When no more violated cuts can be found, branching is performed, and the process is repeated in every node of the branching

tree. The procedure is correct since Benders cuts are globally valid. However, there are some important and challenging implementation details that need to be examined carefully. For example, both fractional and integer points $\mathbf{x}^*$ generate valid Benders cuts, but there is a trade-off between generating these cuts at every fractional point and the overall execution time. These (possibly shallow) cuts generated at fractional points can quickly accumulate and overload the master problem, causing a slow convergence. A common strategy is, therefore, to generate Benders cuts at fractional points at the root node only. Below, we treat other important questions related to the stabilization and degeneracy issues, cut selection, and cut-strengthening techniques.

4.3 Initial cut-loop phase

It is a common practice to solve the relaxed master problem as an LP first (in the so-called cut-loop phase) and then to use the Benders cuts generated during this phase to initialize the MILP model before entering the branch-and-cut process [61, 143]. This has several algorithmic advantages: presolve techniques can help to tighten or fix variable bounds due to the additional information provided by Benders cuts found in this cut-loop phase, and potentially better branching decisions can be made. Moreover, MILP solvers can use these initial Benders cuts to generate additional general-purpose cutting planes (see Section 2.1) to improve the LP-relaxation value at the root node.

Adding all Benders cuts found during this phase may overload the master problem, as many of these cuts can be redundant. Some authors keep only those cuts that are binding at the optimal LP solution instead [143, 325]. That way, the LP optimal value is immediately achieved at the root node of the branch-and-cut tree while potentially redundant cuts are avoided.

4.4 Cut selection

The choice of cuts is very important. Major issues with implementing Benders cuts using Kelley's cutting plane scheme [204] are related to the generation of shallow cuts and slow convergence. At each iteration, one generates one or more cuts that are violated by the current (possibly fractional) solution $\mathbf{x}^*$, and this procedure is repeated. The convergence behavior of this separation loop heavily depends on the strategy for generating the next point to cut. This has been observed by many authors in the past, and we refer to seminal works by [241, 267] and later by [296].

Optimality Cuts It happens very frequently that the dual $\max\{(\mathbf{b} - A\mathbf{x})^\top \mathbf{u} : \mathbf{u} \in F\}$ has multiple optimal solutions. Each of these solutions generates a valid Benders cut, however, not all of them are equally strong. To reduce the number of iterations when separating Benders cuts, one would like to choose an optimal dual solution that yields the "strongest" cut. This has motivated [241] to introduce the notion of *Pareto-Optimal Benders cuts*. A cut corresponding to $\mathbf{u}_1 \in F$ dominates that corresponding to $\mathbf{u}_2 \in F$ if:

$$(\mathbf{b} - A\mathbf{x})^\top \mathbf{u}_1 \geq (\mathbf{b} - A\mathbf{x})^\top \mathbf{u}_2, \quad \forall x \in X.$$

Benders optimality cut (12c) is *Pareto-optimal* if it is not dominated by any other cut.

Magnanti & Wong [241] propose a procedure to generate Pareto-optimal cuts. It is a two-phase approach in which two LPs need to be solved. Let us assume that no feasibility cuts are violated. Let $(\mathbf{x}^*, \theta^*)$ be the optimal solution found at the current iteration. Let $v^* = \Phi(\mathbf{x}^*)$ be the optimal solution value of the Benders subproblem. Among all optimal dual solutions of this subproblem, we now search for the one that generates a non-dominated cut by applying the second phase in which the following LP is solved:

$$MW(\mathbf{x}^*) = \max\{(\mathbf{b} - A\mathbf{x}_0)^\top \mathbf{u} : (\mathbf{b} - A\mathbf{x}^*)^\top \mathbf{u} = v^*, \mathbf{u} \in F\}$$

The point $\mathbf{x}_0$ is called the *core point*. It is assumed to be in the relative interior of $conv(X)$. Such generated cuts are stronger since the optimal dual solution is guaranteed to be Pareto optimal and, hence, non-dominated by other optimal dual points. Thus, in theory, one would need less iterations for the algorithm to converge. Nevertheless, two major drawbacks have been empirically observed in the literature: 1) it might be tricky to find a good core point $\mathbf{x}_0$, and 2) two LPs need to be solved in order to separate a single Benders cut, where the second LP may be more time-consuming and numerically unstable. To circumvent the former issue, Papadakos [267] has shown that $\mathbf{x}_0$ does not have to be a core point and made the separation procedure independent on v^*. Some authors proposed to use the analytic center as a core point instead, see, e.g., [260]. The impact of different core point selection strategies on the overall performance has been empirically investigated in [325]. The trade-off between the execution time needed to find a non-dominated cut and the number of required iterations should not be neglected. It is not guaranteed that this approach outperforms the standard one in which only the first phase is applied. This is why many authors investigated alternative (and possibly computationally more efficient) ways of strengthening Benders cuts, see below.

Feasibility Cuts When the primal Benders subproblem is infeasible, any arbitrary unbounded dual ray can be returned by an LP solver [154]. This may further slow down the convergence and impose additional numerical difficulties. One possible way to overcome this problem is to use *normalization technique* in which the pointed convex cone representing F is intersected with a *normalization hyperplane*. Similar ideas are used for the separation of *disjunctive cuts* [23, 148] and other general-purpose cuts. For the infeasible Benders primal subproblem we introduce a penalty variable $z_0 \geq 0$, measuring the worst violation among the constraints. By minimizing z_0, we ensure that the problem is feasible if $z_0 = 0$. The normalized primal Benders subproblem is given as:

$$\min_{\mathbf{y} \geq \mathbf{0}} \quad \{z_0 : B\mathbf{y} + z_0 \geq \mathbf{b} - A\mathbf{x}^*\} \tag{13}$$

In the dual space, this means that we have intersected the pointed convex cone F with a hyperplane $\mathbf{1}^\top \mathbf{u} = 1$, i.e., we impose that ℓ_1-norm of the dual vector $\mathbf{u}$ must be one. That way, it is guaranteed that an extreme ray is obtained and better numerical stability is achieved.

Fischetti et al. [154] draw a connection between ℓ_1-normalization and the *irreducible infeasible subsystem* (IIS) of the Benders primal suproblem. IIS is an inclusionwise minimal subset of constraints that renders the subproblem infeasible. If any single constraint is removed from IIS, the resulting subsystem becomes feasible. Finding a minimum-cardinality IIS is an NP-hard problem [15]. Using ℓ_1-norm aims to reduce the cardinality of the support of the optimal vertex of (13) and thus allows to heuristically search for the minimum-cardinality IIS (see [169]). The authors point out that it is computationally more efficient to slightly modify the normalization hyperplane as follows. There is no need to consider dual multipliers associated to constraints $B_i^\top \mathbf{y} \geq b_i - A_i^\top \mathbf{x}^*$ for which the A_i-entries are all zero (index i refers to the i-th row of the Benders subproblem). Consequently, the coefficients of the normalization hyperplane can be set to zero for these variables u_i. In principle, there are no theoretical guarantees concerning the strength of such derived normalized cuts. The normalization procedure of [154] is implemented as part of CPLEX automatic Benders implementation, see [69] for more details.

Sometimes, the subproblem structure can be further exploited to derive tailored normalized Benders cuts, see, e.g. [11, 75]. This is the case for subproblems, where e.g., the only source of infeasibility is a single covering constraint of type $B_i^\top \mathbf{y} \geq b_i - A_i^\top \mathbf{x}^*$, for some i. In that case, the Benders subproblem can be restated as

$$\Phi_{-i}(\mathbf{x}^*) = \max_{\mathbf{y} \geq \mathbf{0}} \{B_i^\top \mathbf{y} : B_{-i}\mathbf{y} \geq \mathbf{b}_{-i} - A_{-i}\mathbf{x}^*\}$$

where index "$-i$" stands for the original matrix/vector from which the i-th row is removed. Let $\mathbf{v}_{-i}^p$ be an extreme point of the dual of the above subproblem. The Benders feasibility cuts can then be expressed as:

$$A_i\mathbf{x} + (\mathbf{b} - A_{-i}\mathbf{x})^\top \mathbf{v}_{-i}^p \geq b_i.$$

It is not difficult to see that these cuts result from intersecting the dual pointed cone F with the normalization hyperplane $\mathbf{e}_i^\top \mathbf{u} = 1$.

Facet-defining cuts of [93] and "closest" cuts of [296]. Conforti & Wolsey [93] proposed another two-phase procedure in order to derive a facet-defining Benders cut (feasibility or optimality one). The method requires a core point $\mathbf{x}_0$ and finds a cut on the line segment between $\mathbf{x}_0$ and $\mathbf{x}^*$ (current solution of the master problem), which is valid and further away from $\mathbf{x}_0$. With the help of variable $0 \leq \lambda \leq 1$, the point on the segment is determined as a convex combination $\lambda\mathbf{x}_0 + (1-\lambda)\mathbf{x}^*$. We illustrate how a feasibility cut is derived for a given point $\mathbf{x}^*$, which renders $\Phi(\mathbf{x}^*)$ infeasible. In its primal form, the separation LP is given as:

$$\begin{aligned} \min_{\mathbf{y} \geq \mathbf{0}, 0 \leq \lambda \leq 1} \quad & \lambda \\ \text{s.t.} \quad & B\mathbf{y} + \lambda[A(\mathbf{x}_0 - \mathbf{x}^*)] \geq b - A\mathbf{x}^* \end{aligned}$$

The respective dual (from which the coefficients of the Benders feasibility cut are derived) is then:

$$\max_{\mathbf{u} \geq \mathbf{0}} \quad \{(b - A\mathbf{x}^*)^\top \mathbf{u} : B^\top \mathbf{u} \leq \mathbf{0},\ \mathbf{u}^\top A(\mathbf{x}_0 - \mathbf{x}^*) = 1\}.$$

Solving the above LP is harder and computationally more demanding than using the standard ℓ_1-normalization mentioned above [69]. However, a facet-defining cut or an improper face of the polyhedron can be obtained using this procedure. This normalization technique is also implemented within the CPLEX solver [69].

Recently, Seo et al. [296] proposed a novel cut selection procedure based on fractional programming to find the "closest Benders cut" among all optimality and feasibility cuts. This is a cut that intersects the line segment connecting $\mathbf{x}_0$ and $\mathbf{x}^*$ at the point closest to $\mathbf{x}_0$. The authors show that the resulting cuts are also Pareto optimal and establish a link between their cuts and IIS Benders cuts from [154].

Other Cut Strengthening Techniques Cut strengthening is a highly active area of research, and due to space limitations, we focus here on a few promising recent results from the literature. Strengthened Benders cuts can be obtained by making use of integrality restrictions on $\mathbf{x}$ variables. For example, in [61], the authors propose two ways to do so. The *project-and-cut* approach first projects out $\mathbf{y}$ variables and then uses integrality information on $\mathbf{x}$ to derive strong valid inequalities in the space of $(\mathbf{x}, \theta)$ variables. The *cut-and-project* approach adds valid inequalities (imposed by integrality restrictions on $\mathbf{x}$) directly to the Benders subproblem, after which the variables $\mathbf{y}$ are projected out. The authors conclude that, if there are multiple Benders subproblems and split-cuts are used, stronger Benders cuts and tighter lower bounds are obtained using the cut-and-project approach. The cut-and-project ideas have been also exploited for a more difficult class of two-stage stochastic problems with integer recourse, see, e.g., [160, 326] and further references therein.

Benders cuts can also be derived by looking at the subproblem from the Lagrangian dual perspective. We extend the Benders primal model (10) with auxiliary variables $\mathbf{z}$ and add constraints $\mathbf{z} = \mathbf{x}^*$ to it. Then, for optimal Lagrangian dual multipliers λ^* associated with constraints $\mathbf{z} = \mathbf{x}^*$, and optimal solution $\mathbf{y}^*$ of the Benders subproblem, we obtain the Benders optimality cut (these cuts are derived from subgradients of $\Phi(\mathbf{x}^*)$, see, e.g., [166, 143]):

$$\theta \geq \mathbf{d}^\top \mathbf{y}^* - (\mathbf{x} - \mathbf{x}^*)^\top \lambda^*.$$

Similar reasoning can be applied to derive Benders feasibility cuts.

To strengthen Benders cuts derived from fractional points $\mathbf{x}^*$, *Benders Dual Decomposition* (BDD) is proposed in [276]. The main idea behind BDD consists of adding additional constraints $\mathbf{z} \in X$ into the above model and then relaxing constraints $\mathbf{z} = \mathbf{x}^*$ in Lagrangian fashion (i.e., the subproblem is turned into an MILP). The following Lagrangian subproblem is obtained:

$$\min_{\mathbf{y} \geq \mathbf{0}} \{\mathbf{d}^\top \mathbf{y} - \lambda^\top (\mathbf{z} - \mathbf{x}^*) : B\mathbf{y} \geq \mathbf{b} - A\mathbf{z},\ \mathbf{z} \in X\} \tag{14}$$

If we set $\lambda = \lambda^*$ and find the optimal solution $(\mathbf{y}^*, \mathbf{z}^*)$ of (14), we obtain potentially stronger Benders cuts for fractional $\mathbf{x}^*$ values, see [276]:

$$\theta \geq \mathbf{d}^\top \mathbf{y}^* - (\mathbf{x} - \mathbf{z}^*)^\top \lambda^*.$$

We can further solve the full Lagrangian subproblem

$$\max_{\lambda \geq \mathbf{0}} \min_{\mathbf{y} \geq \mathbf{0}} \{\mathbf{d}^\top \mathbf{y} - \lambda^\top (\mathbf{z} - \mathbf{x}^*) : B\mathbf{y} \geq \mathbf{b} - A\mathbf{z},\ \mathbf{z} \in X\} \tag{15}$$

to obtain the optimal solution $(\tilde{\lambda}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}})$. In that case, according to [276], even stronger *Lagrangian cuts* are obtained:

$$\theta \geq \mathbf{d}^\top \tilde{\mathbf{y}} - (\mathbf{x} - \tilde{\mathbf{z}})^\top \tilde{\lambda}.$$

The procedure of finding $(\tilde{\lambda}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}})$ is computationally demanding, and therefore the authors of [276] propose several heuristic techniques to obtain dual multipliers that will result in stronger Benders cuts. Moreover, if the Lagrangian dual (14) is solved to ϵ-optimality for a given multiplier λ, the following *ϵ-optimal cuts* can be derived:

$$\theta \geq \mathbf{d}^\top \mathbf{y}^* - (\mathbf{x} - \mathbf{z}^*)^\top \lambda - \epsilon$$

where $(\mathbf{y}^*, \mathbf{z}^*)$ represents the ϵ-optimal solution of (14). Recently, more effective ways of generating these Lagrangian cuts (including a cut normalization procedure) have been proposed in [81].

Stabilization We explain a simple-to-implement stabilization strategy, which was originally proposed as a dual stabilization technique for column generation in [37] (see Section 3) and later applied to Benders decomposition in [142]. The method is known as *in-out strategy*. Instead of trying to separate the current LP-optimal point $(\mathbf{x}^*, \theta^*)$, we build a convex combination between this point and a core point $(\mathbf{x}_0, \theta_0)$. The core point is supposed to belong to the relative interior of the projection of the feasible region of (1) onto the variables $(\mathbf{x}, \theta)$. Applying binary search (for several iterations) on the line segment joining these two points, the method shifts the point $(\mathbf{x}^*, \theta^*)$ closer to the feasible region and hence generates a potentially tighter cut. The core point can be obtained by exploiting the specific problem structure or by taking the analytic center instead. The latter can be obtained by, e.g., solving the LP relaxation of the model (1) without the objective and using the barrier algorithm without crossover [69].

Other Acceleration Techniques Many authors have observed that the master problem does not need to be solved to optimality in every iteration of the decomposition process. Instead of using the simplex or interior-point method to find the optimal LP solution, techniques such as constraint programming, metaheuristics, or column generation can be used to solve the relaxed master problem. These techniques can lead to significant speed-ups compared to standard implementation in which the master problem is solved to optimality in each

iteration, see [277] for some recent references. However, to ensure convergence, solving the master problem to optimality (at least in some later iterations) cannot be avoided. Similarly, suboptimal solutions of the Benders subproblem can be used to generate valid cuts, also known as *inexact cuts* [323].

Recent implementations of Benders decomposition incorporate additional advanced acceleration techniques, including the generation of strong cuts, warm-start, and heuristics, or lower bound reinforcing (see [278, 325, 220] and further references therein).

4.5 Separability and subproblem aggregation

When matrix B exhibits a block diagonal structure, one can further decompose the Benders subproblem into K smaller subproblems and introduce K many auxiliary variables $(\theta_1, \ldots, \theta_K)$ to replace the optimal solution value in the objective function (hence, $\theta = \sum_{k=1}^{K} \theta_k$ holds). For each subproblem k, we can then independently add violated feasibility or optimality cuts. The method is known as the *multi-cut* approach, whereas the original one in which a single aggregated cut is added (even when the K subproblems are solved independently) is known as the *single-cut* approach. It is well-known that a trade-off between these two separation paradigms exists. Single-cuts involve fewer variables and produce fewer cuts separated during each cutting plane iteration. Thus, solving the master problem at each cutting plane iteration is faster than solving the problem via the multi-cuts implementation. However, optimality cuts in their single-cut form can only be added when the whole set of subproblems is feasible. Moreover, at any given iteration, single cuts are weaker than their multi-cut counterpart, so the multi-cuts implementation may converge in a fewer number of iterations. In the multi-cut context, optimality cuts can be generated for a subset of subproblems, even though some other subproblems remain infeasible. In general, however, multi-cuts add many more cuts to the master problem, thus making it larger and slower to solve.

This trade-off is addressed in the context of two-stage stochastic optimization in [308, 36, 110, 280] by exploiting scenario aggregation techniques. Specifically, Crainic et al. [110] proposed to retain some important scenarios in the master problem and to generate some artificial ones in order to derive strengthening valid inequalities. The method is known as *partial Benders decomposition*. The artificial scenarios are based on scenario aggregation, so the authors use them as a cut-strengthening technique but still derive multi-cuts for scenarios whose second-stage variables have been projected out.

It is only with the recent introduction of the *Benders adaptive-cut* approach by Ramirez-Pico et al. [280] that necessary and sufficient conditions are provided, guaranteeing that certain scenario-aggregated Benders cuts can provide an optimal solution of the starting problem. This idea of aggregating scenarios was first explored to solve the extended formulation of two-stage stochastic programs. Dimension of such large-scale MILP can be drastically reduced if the scenario set is partitioned and one representative scenario is used per each element of the partition. Such partition can then be iteratively refined to improve the obtained lower and upper bounds until the two bounds meet. This is the main idea behind the (generalized) *adaptive partition method*, APM and GAPM, proposed in [304] and [281], respectively. Following the theory of the GAPM, the idea of aggregating scenarios in the Benders decomposition context is formalized in [280]. The essential idea behind their *Benders adaptive-cuts* is to aggregate all scenarios to obtain a smaller and easier master problem, and then iteratively solve the subproblems and disaggregate the scenarios into subsets sharing, e.g., the same dual optimal solutions. The authors provide sufficient conditions under which a given set of scenarios can remain aggregated without sacrificing the optimality. The advantages of both the single-cut and multi-cut approach are integrated in this new adaptive framework.

4.6 Logic-based Benders decomposition

The original Benders decomposition approach has been extended by Hooker [185] by replacing the LP duality by the *inference duality*, in which a strongest possible bound from the Benders subproblem is logically deduced from its constraint set. This concept allows us to extend the general idea of Benders decomposition to more complex subproblems that may contain discrete variables or non-linear objective functions or constraints. Hence, subproblems can be solved using different types of solvers, among which constraint programming has shown significant speed-ups for certain problem classes like scheduling and planning [188]. Some early ideas are described in [187]. Hooker [186] provides a classification of logic-based Benders cuts, and we emphasize some of the most important families below. For simplicity, we will assume that variables $\mathbf{x}$ are binary and belong to the master problem and that variables $\mathbf{y}$ are projected out. However, the reader should keep in mind that the logic-based Benders decomposition can also be applied when some (or all) discrete variables are sent to the subproblem(s).

- *Nogood cuts* are used to exclude the most recent solution, say $\mathbf{x}^*$, of the master problem. Let $J_1(\mathbf{x}^*)$ be the set of indices i for which $x_i^* = 1$, and $J_0(\mathbf{x}^*)$ be the set of indices i for which $x_i^* = 0$ holds. Nogood cuts can be given in the form of feasibility cuts (in case $\mathbf{x}^*$ is infeasible, ensuring that at least one variable

from $\mathbf{x}^*$ must change its value)

$$\sum_{i \in J_0(\mathbf{x}^*)} x_i + \sum_{i \in J_1(\mathbf{x}^*)} (1 - x_i) \geq 1$$

or optimality cuts

$$\theta \geq \underline{\theta} + (\Phi(\mathbf{x}^*) - \underline{\theta}) \left[\sum_{i \in J_1(\mathbf{x}^*)} (x_i - 1) - \sum_{i \in J_0(\mathbf{x}^*)} x_i + 1 \right],$$

where $\underline{\theta}$ is a global lower bound of the Benders subproblem. The optimality cut imposes a lower bound on θ when $\mathbf{x} = \mathbf{x}^*$, but no lower bound otherwise. We notice that optimality cuts resemble integer L-shaped cuts for two-stage stochastic optimization with integer recourse and binary first-stage variables [222]. Nogood cuts are known to be very weak, as they exclude one point at a time, and many enhancements are possible, see below.

- *Monotone nogood cuts* can exclude a class of solutions that dominate the most recent solution. They are applied when the function $\Phi(\mathbf{x})$ is monotone in $\mathbf{x}$, i.e., when $\mathbf{x} \geq \mathbf{x}'$ implies $\Phi(\mathbf{x}) \geq \Phi(\mathbf{x}')$ (we assume that if $\mathbf{x}$ is infeasible, then $\Phi(\mathbf{x}) = \infty$). In that case, in the above cuts, we can set $J_0(x^*) := \emptyset$, and hence not a single point $\mathbf{x}^*$, but all other binary points $\mathbf{x}$ such that $\mathbf{x} \geq \mathbf{x}^*$, will be also cut off using the respective feasibility or optimality cut.

- *Irreducible monotone nogood cuts* are obtained by further reducing the size of the set $J_1(\mathbf{x}^*)$ defined above. The idea is to exclude from $J_1(\mathbf{x}^*)$ those variables x_i with no impact on $\Phi(\mathbf{x}^*)$. Let $I_1(\mathbf{x}^*) \subset J_1(\mathbf{x}^*)$ be a subset of indices, where we have removed some of these variables. The feasibility cut

  $$\sum_{i \in I_1(\mathbf{x}^*)} (1 - x_i) \geq 1$$

  is called *irreducible* if no further variable can be removed from $I_1(\mathbf{x}^*)$ without rendering the cut invalid (similar reasoning applies to respective optimality cuts). Finding an irreducible subset is usually a difficult task, as it requires finding an inclusionwise minimal subset of master variables that cause infeasibility of the subproblem (or that contribute to the value of $\Phi(\mathbf{x}^*)$, in case of optimality cuts). To find such subsets, a greedy algorithm that tries to sequentially flip some of the variables x_i from 1 to 0 has been proposed in [188]. The algorithm is a heuristic with no guarantee that the resulting set will be irreducible. Other algorithms that guarantee finding an irreducible set are based on a deletion filter approach (see, e.g., [219]), an additive method (that builds an irreducible subset from scratch); a combination of the former two, or a combination of the delete filter with a binary search procedure like Depth-First Binary Search [202] and QuickXplain [199]. A detailed computational study regarding these strengthening techniques has been conducted in [292]. The authors conclude that the greedy approach is outperformed by the other techniques and that, in general, the deletion filter and the Depth-First Binary Search solve a lower number of subproblems and, therefore, perform slightly better in practice.

- *Combinatorial Benders cuts*, introduced in [88], are a special case of irreducible nogood cuts specially designed for MILPs.

- Other relevant families of cuts include *analytical cuts* that allow excluding a class of solutions by exploiting the most recent solution along with the specific problem structure. We observe that in a noogood cut, a tight bound on θ is only implied when each variable in $I_1(\mathbf{x}^*)$ has the value 1. The coefficients of analytical cuts overcome this drawback by taking into consideration how changing the values of the decision variables affects the objective value. However, they need to be designed specifically for each problem under consideration. These cuts can be used along with the (irreducible/monotone) optimality cuts or as an alternative.

An overview of the theoretical developments and applications in the last two decades can be found in the recent book by Hooker [186].

4.7 Automatic Benders decomposition in modern solvers

CPLEX and SCIP are two solvers in which the user can plug in a compact MILP in the form of (1), and ask solvers to automatically apply Benders decomposition (the user can define the variables of the master and the subproblem(s), or leave it to the solver to recognize the structure and decompose the system automatically). Implementation details crucial for the success of the method for CPLEX and SCIP can be found in [69] and [242], respectively. Their computational results show that the automatic Benders decomposition outperforms the default branch-and-cut on models amenable to decomposition.

As to the logic-based Benders decomposition, despite its flexibility to handle various types of subproblems, the method usually requires strengthened logic-based Benders cuts to be designed specifically for every class of problems. A generic solver `Nutmeg` that hybridizes MILP and constraint programming techniques and automatically generates logic-based Benders cuts within a branch-and-cut framework has been proposed in [219]. The obtained results are promising for classes of problems that can be naturally decomposed into MILP master and CP subproblems. However, developing a generic cut-strengthening approach that can be applied to arbitrary problems remains a challenge.

5 Conclusions and perspectives

We are closing the article by first summarizing the major trends and short-term perspectives for MILP computation. We then turn our attention to (rather long-term) challenges and opportunities for the MILP community, reflecting on gaps between the current state of research and real-world needs.

5.1 Current trends and perspectives

Learning and MILP computation In the past ten years, we have witnessed many new ideas that have tried to integrate ML with MILPs. This trend is ongoing and represents one of the most active areas of MILP research. An overview of recent techniques used to integrate ML and combinatorial optimization can be found in Bengio et al. [42]. According to the authors, one can classify the current techniques into three groups: (1) *end to end learning* (where ML is trained to output solutions directly from the input data); (2) *learning to configure algorithms* (where ML is trained to provide better parameters, input data, or models for MILPs) and (3) *ML alongside optimization* (similar to (2), but ML is repeatedly used and trained during the execution of the optimization algorithm). The first approach completely replaces black-box MILP solvers with their ML counterparts, so in the long run, it may pose a threat to the MILP community. However, the current implementations have some serious limitations (e.g., ML has to be intensively trained for every specific problem, and ML approaches remain heuristic in nature). In the context of MILP computation, the latter two techniques have proven to be very successful. Among them, we highlight:

- learning decision strategies within MILP solvers: some promising directions for improving the computational performance of MILP solvers have recently been proposed. For example, despite the variety of different types of cutting planes available today, there is still a lack of understanding regarding which and how many cuts should be generated in each iteration, which cuts should be kept or removed from the master problem, or which separation algorithms (heuristic or exact) should be used to generate them. Learning to select and generate cutting planes is indeed one of the trending directions of ongoing and future research [50, 129, 307]. Other examples include learning to branch [205], learning to select columns [252], or learning to use primal heuristics [82, 206]. Similarly, deciding whether a decomposition should be used and detecting the right submatrices to use for the decomposition can also be learned, as demonstrated in [45, 215].
- learning MILP-based heuristics to obtain high-quality solutions: while the former approaches are essentially integrated as part of MILP solving methods, this class uses MILP solvers as a black box. An example is given in [301], where a technique is proposed in order to learn how to apply and guide LNS for generic MILPs.

These are just a few examples of highly relevant questions that need further attention from the MILP community. Note, however, that at the time we are writing this survey, no ML-based technique is included in the most crucial parts of the best solvers (i.e., branching or cut selection). To facilitate a comparison of ML-guided MILP solution algorithms, a common repository containing distributions of MILPs along with standardized test sets for ML approaches has been recently proposed in [190].

Hybridization CP/AI/MILP Modern solvers already use Constraint Programming (CP) and SAT-solver-inspired techniques for node preprocessing and branching. Researchers have also designed hybrid methods, generally based on decompositions. A survey on the integration of CP and MILP can be found in [189]. The authors identify four main possible types of integration: using LP relaxation to help CP propagation, using specialized algorithms from the OR field (matching, dynamic programming) for domain filtering, decomposition (typically Dantzig-Wolfe and combinatorial Benders, see Sections 3 and 4), and decision diagrams (see section 3). Using CP to solve subproblems in column generation methods [122] or Benders' decomposition [195] is a natural choice when the subproblem is highly combinatorial. CP also allows modeling the pricing subproblem with a formal language [109]. Note that decision diagrams, initially proposed in the field of AI and CP and initially leveraged for solving large dynamic programs, are now also used to produce MILP formulations (see, e.g., [216]).

Deep neural networks (DNN) based on Rectified Linear Unit (ReLU) activation functions can also be analyzed from the MILP perspective. Training of a *feedforward rectifier network* with ReLU activation functions is equivalent to performing a piecewise linear regression. This creates an interesting link between polyhedral theory and neural networks [192]. This link allows using MILP techniques for training, verifying or reducing DNNs. Moreover, it allows for tackling other important questions, such as the robustness of DNNs, their susceptibility to adversarial attacks, or the integration of models learned by DNNs into decision-making models. The survey of Huchette et al. [192] provides an excellent overview of the recent developments and clearly demonstrates the relevance of the topic for both AI and MILP communities.

Overall, we find that there is still unexplored potential in combining the best of the three worlds (CP, AI and MILPs), and we expect to see more interesting developments in the near future.

MILPs for decision making Many difficult decision-making problems are of a non-deterministic (or online) nature. Some of them involve multiple objectives or multiple decision-makers as well. The recent progress in solving difficult and large-scale MILPs using modern solvers (see Section 1) opens plenty of possibilities to extend the scope of applications and to model and solve optimally some decision-making problems that were out of reach only a decade before. We highlight some of the interesting and promising ongoing developments.

When it comes to optimization under uncertainty, we distinguish between stochastic and robust optimization. Discrete two-stage and multi-stage stochastic optimization problems can be modeled as large-scale MILPs using the sample average approximation technique [209]. Usually, problem-tailored decomposition approaches are needed to cope with the problem size. However, SCIP and CPLEX also provide automatic decomposition tools with which the problems can be attacked (see [69] and [242], respectively). The application of these black-box tools is currently limited to the problems that satisfy the basic assumptions specific to each decomposition approach. However, many challenging applications do not necessarily satisfy these assumptions. For example, two-stage or multi-stage stochastic optimization problems with integer (or non-linear) recourse are still highly challenging to solve, and state-of-the-art methods naturally rely on decomposition methods [81, 115]. Similarly, the field of robust optimization is flourishing, and MILP-based decomposition approaches also represent state-of-the-art, see, e.g., [22, 63]. All these algorithms are highly specialized, and in principle, the community lacks more powerful solvers capable of tackling more general settings of multi-stage discrete optimization problems under uncertainty.

When it comes to involving multiple decision-makers, MILP models can be integrated into the basic concepts of game theory as well. For sequential decision-making, for example, Stackelberg games (also known as leader-follower games) are solved using bilevel optimization, and the field is growing rapidly. In the last decade, we have seen significant algorithmic advances for problems where the optimization problems of players can be modeled as MILPs [208, 35]. Techniques for solving such bilevel problems also rely on reformulations, decompositions, intersection or disjunctive cuts, and their integration into branch-and-cut trees (see the recent survey on the computational aspect of bilevel optimization [208]). When decision-makers are playing simultaneously, *integer programming games* allow to compute Nash equilibria for non-cooperative non-convex games, where each player solves a MILP [211, 80].

For solving MILPs with multiple objectives, the existing MILP-based approaches can be divided into (1) those that work in the space of objective function values (see, e.g., [64, 65]) and (2) those that work in the space of decision variables and rely on generalizations of the branch-and-bound technique (see, e.g., [155] and further references therein). We believe that scalability achieved by MILPs will also perpetuate the further development of more efficient computational tools for supporting multi-objective decision-making.

Intermediate modeling tools and algorithm libraries Our experience with decomposition methods has also convinced us that the community needs intermediate modeling tools. For example, testing a new idea to solve exactly a classical variant of the vehicle routing problem with a column-generation method necessitates implementing a column-generation process, branching tree management, stabilization, primal heuristics, efficient labeling techniques, and dedicated cut generation. It is clearly out of reach of a single researcher unless they have access to highly skilled engineers who can spend months on a project to reproduce the state of the art. Making the best implementations available would help advanced developers, but these codes are generally hard to understand, do not always have documentation, and necessitate advanced programming skills to modify if one wants to adapt them to a new problem. This calls for intermediate modeling tools with which a practitioner can express a problem using a high-level formalism and have access to these advanced algorithms. Several libraries pave the way for these solvers. SCIP [67] already includes many state-of-the-art features and allows the plugging of user-defined algorithms. In [305], the authors propose a generic branch and cut for bilevel optimization with a computer implementation. The black-box VRPSolver [271] (see also VRPSolverEasy, `pypi.org/project/VRPSolverEasy/`) also allows modeling a large class of optimization problems as VRP problems and gives access to the most recent advances in branch-and-cut-and-price for VRP.

5.2 Challenges and opportunities

Applications of MILP techniques Companies nowadays widely use AI techniques to extract value from big data. However, the role of MILPs in this process cannot be overlooked, as the explainability and interpretability of the models employed and the results obtained still remain relevant for some of the applications. This gives MILPs a competitive advantage over black-box AI techniques, as the end users can much more easily understand and interpret the obtained results in relation to the constraints imposed on the MILP model. New and challenging big-data applications for MILPs can be found across different sectors. In the telecommunication industry, for example, new topics include the development and deployment of programmable networks, placement of virtual functions, cloud solutions, or energy optimization. In transportation and logistics, many studies have shifted from traditional cost minimization to CO2 emission reduction. They focus on last-mile optimization with modeling challenges related to new modes of transportation (including drones, e-vehicles, autonomous vehicles, or occasional drivers). Healthcare and cyber-security provide additional sources of demanding applications for MILPs.

Modern or future **computer architectures** will also be an opportunity to push further the advances in MILP-based algorithms. State-of-the-art methods for solving LPs are based on simplex [113] and barrier algorithms [203, 282] (see, e.g., [60] for a brief history on methods for mixed-integer linear programming up to 2012). Motivated by the success of GPU architecture for machine learning, researchers are nowadays investigating how to unlock the potential of GPU architecture and distributed systems for solving large-scale LPs. Matrix factorizations required by these methods represent a major bottleneck for running them on modern architectures. As an alternative, first-order methods are the current method of choice (see, e.g., [20]), and an overview of the most recent developments can be found in [238]. Further challenges regarding how to take advantage of this architecture to solve large-scale MILPs still need to be addressed.

Research is being conducted on developing **quantum computing** algorithms for solving 0/1 ILPs [250]. An arbitrary 0/1 ILP can be transformed into a Quadratic Unconstrained Binary Optimization (QUBO) problem using penalty techniques [171] and then solved approximately on the quantum processing unit by D-Wave using Quantum Annealing (QA). A recent experimental comparison [197] between QA and exact solvers (based on digital computing) has been made on a large benchmark of QUBO instances. The authors conclude: "*It may well be (and we hope) that the exciting new quantum computer technology will make leaps in the future, but in our experiments we have certainly not observed superior performance of quantum annealing in comparison to "classical" methods.*" For solving MILPs, the development of *hybrid quantum–classical* algorithms has been recognized as a promising research direction [1]. For example, decomposition approaches based on Dantzig-Wolfe, Lagrangian or Benders decomposition, or problem reformulation techniques, could provide some good opportunities to simultaneously exploit classical and quantum resources.

Interpretability is or will be a requirement of many decision-making tools. It is well-known that recommendations resulting from ML algorithms suffer from a lack of explanations and transparency. This has been recognized as an opportunity for MILP community. Many recent studies showed that MILPs could be quite useful for developing tools to better interpret the results of black-box ML models or even to design better interpretable models firsthand (see [54, 78] and further references therein). Recently, [243] proposed to use robust optimization to provide counterfactual explanations (i.e., to identify the smallest change in personal data that would lead to a desired model outcome) for many ML models, including logistic regression, decision trees, random forests, and neural networks. On the other hand, interpretability issues can be a challenge for MILP as well. Although model-based methods can be checked by OR experts, gaining trust from end-users is an issue for mathematical programming-based solvers based on complex algorithms and mathematical results. Indeed, due to the lack of transparency in the optimal solutions, the end-users may also perceive the MILP solvers as black-boxes. First works towards the better interpretability objective for MILPs [226] leverage tools based on reoptimization and post-optimal sensitivity analysis to propose interactive tools, where the user can question the solution produced by the system and evaluate alternative choices (counterfactual explanations). For the counterfactual explanations of LPs, see the recent work in [217].

An increasingly important matter in the scientific community is the **reproducibility** of experiments. The best results obtained for solving MILP are obtained using black-box commercial solvers. This has two disadvantages: these solvers are not available to everyone, and it is sometimes difficult to understand our algorithms' behavior fully. Improving open-source solvers would bring our community to the next level of reproducibility. Using open data and well-documented open-source implementations will soon become mandatory for publishing in high-quality journals.

We expect **generative AI** to be able to produce MILP models from a textual description of the problem or examples of data and solutions. Some early promising results have been recently reported in [10]. These generative AI approaches may also be able to help produce efficient formulations from naive ones. Although it may not impact the work of MILP experts, this may change how end-users can access optimization tools. This may also change the way we teach MILP modeling and trigger the need for additional formal tools to check the validity of these models.

Overall, we conclude that we are entering exciting and, at the same time, highly challenging times for the MILP community. The achieved scalability of the MILP solvers and their modeling power are still not sufficiently communicated to the end users. Many companies just follow the tide and simply turn to black-box AI solutions, which are known for their lack of explainability and interpretability and for spending huge amounts of computing resources for the training of their models. On the brighter side, we expect that further hybridizations between AI/ML/CP and MILPs, notably in using various decomposition techniques, will combine the best of the worlds and will be unavoidable as we are facing the development of new architectures and the rise of quantum computing.

References

[1] A. Abbas, A. Ambainis, B. Augustino, A. Bärtschi, H. Buhrman, C. Coffrin, G. Cortiana, V. Dunjko, D. J. Egger, B. G. Elmegreen, N. Franco, F. Fratini, B. Fuller, J. Gacon, C. Gonciulea, S. Gribling, S. Gupta, S. Hadfield, R. Heese, G. Kircher, T. Kleinert, T. Koch, G. Korpas, S. Lenk, J. Marecek, V. Markov, G. Mazzola, S. Mensa, N. Mohseni, G. Nannicini, C. O'Meara, E. P. Tapia, S. Pokutta, M. Proissl, P. Rebentrost, E. Sahin, B. C. B. Symons, S. Tornow, V. Valls, S. Woerner, M. L. Wolf-Bauwens, J. Yard, S. Yarkoni, D. Zechiel, S. Zhuk, and C. Zoufal. Quantum optimization: Potential, challenges, and the path forward. `https://arxiv.org/abs/2312.02279`, 2023.

[2] T. Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, 2007.

[3] T. Achterberg and T. Berthold. Hybrid Branching. In W.-J. van Hoeve and J. N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Notes in Computer Science, pages 309–311, Berlin, Heidelberg, 2009. Springer.

[4] T. Achterberg, T. Berthold, and G. Hendel. Rounding and Propagation Heuristics for Mixed Integer Programming. In D. Klatte, H.-J. Lüthi, and K. Schmedders, editors, *Operations Research Proceedings 2011*, Operations Research Proceedings, pages 71–76, Berlin, Heidelberg, 2012. Springer.

[5] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger. Presolve Reductions in Mixed Integer Programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020.

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

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

[8] T. Achterberg, A. Sabharwal, and H. Samulowitz. Stronger Inference through Implied Literals from Conflicts and Knapsack Covers. In C. Gomes and M. Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Notes in Computer Science, pages 1–11, Berlin, Heidelberg, 2013. Springer.

[9] T. Achterberg and R. Wunderling. Mixed Integer Programming: Analyzing 12 Years of Progress. In M. Jünger and G. Reinelt, editors, *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*, pages 449–481. Springer, Berlin, Heidelberg, 2013.

[10] A. AhmadiTeshnizi, W. Gao, and M. Udell. OptiMUS: Optimization Modeling Using MIP Solvers and large language models. `doi.org/10.48550/arXiv.2310.06116`, 2023.

[11] L. Alfandari, I. Ljubić, and M. de Melo da Silva. A tailored Benders decomposition approach for last-mile delivery with autonomous robots. *European Journal of Operational Research*, 299(2):510–525, 2022.

[12] A. M. Alvarez. *Computational and Theoretical Synergies between Linear Optimization and Supervised Machine Learning*. PhD thesis, Université de Liège, 2016.

[13] A. M. Alvarez, Q. Louveaux, and L. Wehenkel. A Machine Learning-Based Approximation of Strong Branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

[14] C. Alves, F. Clautiaux, J. Valerio De Carvalho, and J. Rietz. *Dual-Feasible Functions for Integer Programming and Combinatorial Optimization*. EURO Advanced Tutorials on Operational Research. Springer International Publishing, Cham, 2016.

[15] E. Amaldi, M. E. Pfetsch, and L. E. T. Jr. On the maximum feasible subsystem problem, iiss and iis-hypergraphs. *Mathematical Programming*, 95(3):533–554, 2003.

[16] K. Andersen, G. Cornuéjols, and Y. Li. Split closure and intersection cuts. *Mathematical Programming*, 102(3):457–493, 2005.

[17] G. Andreello, A. Caprara, and M. Fischetti. Embedding $\{0, \frac{1}{2}\}$-Cuts in a Branch-and-Cut Framework: A Computational Study. *INFORMS Journal on Computing*, 19(2):229–238, 2007.

[18] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding cuts in the TSP (a preliminary report), 1995.

[19] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem*. Princeton University Press, Princeton, 2011.

[20] D. L. Applegate, O. Hinder, H. Lu, and M. Lubin. Faster first-order primal-dual methods for linear programming using restarts and sharpness. *Mathematical Programming*, 201(1):133–184, 2023.

[21] C. Archetti and I. Ljubić. Comparison of formulations for the inventory routing problem. *European Journal of Operational Research*, 303(3):997–1008, 2022.

[22] A. N. Arslan and B. Detienne. Decomposition-Based Approaches for a Class of Two-Stage Robust Binary Optimization Problems. *INFORMS Journal on Computing*, 34(2):857–871, 2022.

[23] E. Balas. A modified lift-and-project procedure. *Mathematical Programming*, 79:19–31, 1997.

[24] E. Balas. *Disjunctive programming*. Springer, 2018.

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

[26] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19(1):1–9, 1996.

[27] E. Balas and T. Kis. On the relationship between standard intersection cuts, lift-and-project cuts, and generalized intersection cuts. *Mathematical Programming*, 160(1-2):85–114, 2016.

[28] E. Balas and M. Perregaard. Lift-and-project for Mixed 0–1 programming: Recent progress. *Discrete Applied Mathematics*, 123(1):129–154, 2002.

[29] E. Balas and A. Saxena. Optimizing over the split closure. *Mathematical Programming*, 113(2):219–240, 2008.

[30] E. Balas, S. Schmieta, and C. Wallace. Pivot and shift - a mixed integer programming heuristic. *Discrete Optimization*, 1(1):3–12, 2004.

[31] R. Baldacci and A. Mingozzi. A unified exact method for solving different classes of vehicle routing problems. *Mathematical Programming*, 120(2):347–380, 2009.

[32] R. Baldacci, A. Mingozzi, and R. Roberti. New Route Relaxation and Pricing Strategies for the Vehicle Routing Problem. *Operations Research*, 59(5):1269–1283, 2011.

[33] F. Barahona and R. Anbil. The volume algorithm: Producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399, 2000.

[34] E. M. L. Beale and J. A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. *OR*, 69(447-454):99, 1970.

[35] Y. Beck, I. Ljubić, and M. Schmidt. A survey on bilevel optimization under uncertainty. *European Journal of Operational Research*, 311(2):401–426, 2023.

[36] C. Beltran-Royo. Fast scenario reduction by conditional scenarios in two-stage stochastic MILP problems. *Optimization Methods and Software*, pages 1–22, 2019.

[37] W. Ben-Ameur and J. Neto. Acceleration of cutting-plane and column generation algorithms: Applications to network design. *Networks*, 49(1):3–17, 2007.

[38] H. Ben Amor, J. Desrosiers, and J. M. Valério De Carvalho. Dual-Optimal Inequalities for Stabilized Column Generation. *Operations Research*, 54(3):454–463, 2006.

[39] H. M. T. Ben Amor, J. Desrosiers, and A. Frangioni. On the choice of explicit stabilizing terms in column generation. *Discrete Applied Mathematics*, 157(6):1167–1184, 2009.

[40] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

[41] P. Bendotti, P. Fouilhoux, and C. Rottner. Symmetry-breaking inequalities for ILP with structured sub-symmetry. *Mathematical Programming*, 183(1):61–103, 2020.

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

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

[44] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. Hooker. *Decision Diagrams for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer International Publishing, Cham, 2016.

[45] M. Bergner, A. Caprara, A. Ceselli, F. Furini, M. E. Lübbecke, E. Malaguti, and E. Traversi. Automatic Dantzig–Wolfe reformulation of mixed integer programs. *Mathematical Programming*, 149(1):391–424, 2015.

[46] L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63–76, 2007.

[47] T. Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013.

[48] T. Berthold. RENS. *Mathematical Programming Computation*, 6(1):33–54, 2014.

[49] T. Berthold. A computational study of primal heuristics inside an MI(NL)P solver. *Journal of Global Optimization*, 70(1):189–206, 2018.

[50] T. Berthold, M. Francobaldi, and G. Hendel. Learning to use local cuts. `https://arxiv.org/abs/2206.11618`, 2022.

[51] T. Berthold, G. Hendel, and D. Salvagnin. Transferring Information Across Restarts in MIP. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 19th International Conference, CPAIOR 2022, Los Angeles, CA, USA, June 20-23, 2022, Proceedings*, pages 24–33, Berlin, Heidelberg, 2022. Springer-Verlag.

[52] T. Berthold, A. Lodi, and D. Salvagnin. Ten years of feasibility pump, and counting. *EURO Journal of Computational Optimization*, 7(1):1–14, 2019.

[53] T. Berthold, M. Perregaard, and C. Mészáros. Four good reasons to use an interior point solver within a MIP solver. In N. Kliewer, J. F. Ehmke, and R. Borndörfer, editors, *Operations Research Proceedings 2017, Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Freie Universiät Berlin, Germany, September 6-8, 2017*, Operations Research Proceedings, pages 159–164. Springer, 2017.

[54] D. Bertsimas, A. Orfanoudaki, and H. M. Wiberg. Interpretable clustering: an optimization approach. *Machine Learning*, 110(1):89–138, 2021.

[55] D. Bienstock and M. Zuckerberg. Solving LP relaxations of large-scale precedence constrained problems. In F. Eisenbrand and F. B. Shepherd, editors, *Integer Programming and Combinatorial Optimization*, pages 1–14, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[56] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer-Verlag, New York, NY, USA, 1997.

[57] R. Bixby. MIE Distinguished Seminar Series. Optimization: Past, Present, Future with Dr. Robert Bixby. `https://www.youtube.com/watch?v=_R8-nt5NyiE`, 2021.

[58] R. Bixby and E. Rothberg. 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, 2007.

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

[60] R. E. Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, 2012:107–121, 2012.

[61] M. Bodur, S. Dash, O. Günlük, and J. Luedtke. Strengthened Benders cuts for stochastic integer programs with continuous recourse. *INFORMS Journal on Computing*, 29(1):77–91, 2017.

[62] M. Bodur and J. R. Luedtke. Mixed-integer rounding enhanced Benders decomposition for multiclass service-system staffing and scheduling with arrival rate uncertainty. *Management Science*, 63(7):2073–2091, 2017.

[63] M. Bodur and J. R. Luedtke. Two-stage linear decision rules for multi-stage stochastic programming. *Mathematical Programming*, 191(1):347–380, 2022.

[64] N. Boland, H. Charkhgard, and M. W. P. Savelsbergh. A criterion space search algorithm for biobjective mixed integer programming: The triangle splitting method. *INFORMS Journal on Computing*, 27(4):597–618, 2015.

[65] N. Boland, H. Charkhgard, and M. W. P. Savelsbergh. The quadrant shrinking method: A simple and efficient algorithm for solving tri-objective integer programs. *European Journal of Operational Research*, 260(3):873–885, 2017.

[66] N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. The Continuous-Time Service Network Design Problem. *Operations Research*, 65(5):1303–1321, 2017.

[67] S. Bolusani, M. Besançon, K. Bestuzheva, A. Chmiela, J. Dionísio, T. Donkiewicz, J. van Doornmalen, L. Eifler, M. Ghannam, A. Gleixner, C. Graczyk, K. Halbig, I. Hedtke, A. Hoen, C. Hojny, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, J. Manns, G. Mexi, E. Mühmer, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, D. Weninger, and L. Xu. The SCIP Optimization Suite 9.0. ZIB-Report 24-02-29, Zuse Institute Berlin, February 2024.

[68] P. Bonami. On optimizing over lift-and-project closures. *Mathematical Programming Computation*, 4(2):151–179, 2012.

[69] P. Bonami, D. Salvagnin, and A. Tramontani. Implementing Automatic Benders Decomposition in a Modern MIP Solver. In D. Bienstock and G. Zambelli, editors, *Integer Programming and Combinatorial Optimization*, pages 78–90, Cham, 2020. Springer International Publishing.

[70] M. A. Boschetti, A. N. Letchford, and V. Maniezzo. Matheuristics: survey and synthesis. *International Transactions in Operational Research*, 30(6):2840–2866, 2023.

[71] M. A. Boschetti and V. Maniezzo. Matheuristics: Using mathematics for heuristic design. *4OR*, 20(2):173–208, 2022.

[72] F. Brandão and J. P. Pedroso. Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69:56–67, 2016.

[73] O. Briant, C. Lemaréchal, Ph. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of bundle and classical column generation. *Mathematical Programming*, 113(2):299–344, 2008.

[74] S. S. Brito and H. G. Santos. Preprocessing and cutting planes with conflict graphs. *Computers & Operations Research*, 128:105176, 2021.

[75] A. Calamita, I. Ljubić, and L. Palagi. Benders decomposition for congested partial set covering location with uncertain demand. `https://arxiv.org/abs/2401.12625`, 2024.

[76] A. Caprara and M. Fischetti. {0, 1/2}-Chvátal-Gomory cuts. *Mathematical Programming*, 74(3):221–235, 1996.

[77] A. Caprara and A. N. Letchford. On the separation of split cuts and related inequalities. *Mathematical Programming*, 94(2):279–294, 2003.

[78] E. Carrizosa, K. Kurishchenko, A. Marín, and D. Romero Morales. On clustering and interpreting with rules by means of mathematical optimization. *Computers & Operations Research*, 154:106180, 2023.

[79] R. Carvajal, S. Ahmed, G. Nemhauser, K. Furman, V. Goel, and Y. Shao. Using diversification, communication and parallelism to solve mixed-integer linear programs. *Operations Research Letters*, 42(2):186–189, 2014.

[80] M. Carvalho, G. Dragotto, A. Lodi, and S. Sankaranarayanan. Integer Programming Games: A Gentle Computational Overview. In *Tutorials in Operations Research: Advancing the Frontiers of OR/MS: From Methodologies to Applications*, INFORMS TutORials in Operations Research, chapter 2, pages 31–51. INFORMS, 2023.

[81] R. Chen and J. R. Luedtke. On Generating Lagrangian Cuts for Two-Stage Stochastic Integer Programs. *INFORMS Journal on Computing*, 34(4):2332–2349, 2022.

[82] A. Chmiela, E. Khalil, A. Gleixner, A. Lodi, and S. Pokutta. Learning to schedule heuristics in branch and bound. *Advances in Neural Information Processing Systems*, 34:24235–24246, 2021.

[83] N. Christofides, A. Mingozzi, and P. Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164, 1981.

[84] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(4):305–337, 1973.

[85] F. Clautiaux, C. Alves, J. Valério de Carvalho, and J. Rietz. New Stabilization Procedures for the Cutting Stock Problem. *INFORMS Journal on Computing*, 23(4):530–545, 2011.

[86] F. Clautiaux, S. Hanafi, R. Macedo, M.-É. Voge, and C. Alves. Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints. *European Journal of Operational Research*, 258(2):467–477, 2017.

[87] F. Clautiaux, R. Sadykov, F. Vanderbeck, and Q. Viaud. Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem. *Discrete Optimization*, 29:18–44, 2018.

[88] G. Codato and M. Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

[89] COIN-OR. Computational Infrastructure for Operations Research, 2024. Accessed: 2024-06-30.

[90] M. Conforti, G. Cornuéjols, and G. Zambelli. Extended formulations in combinatorial optimization. *4OR*, 8(1):1–48, 2010.

[91] M. Conforti, G. Cornuéjols, and G. Zambelli. Polyhedral approaches to mixed integer linear programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 343–385. Springer, 2010.

[92] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer programming*. Springer, 2014.

[93] M. Conforti and L. A. Wolsey. "Facet" separation with one linear program. *Mathematical Programming*, 178(1–2):361–380, 2019.

[94] S. Coniglio, F. Furini, and I. Ljubić. Submodular maximization of concave utility functions composed with a set-union operator with applications to maximal covering location problems. *Mathematical Programming*, 196(1):9–56, 2022.

[95] S. Coniglio and M. Tieves. On the Generation of Cutting Planes which Maximize the Bound Improvement. In E. Bampis, editor, *Experimental Algorithms*, volume 9125, pages 97–109. Springer International Publishing, Cham, 2015.

[96] C. Contardo, A. Lodi, and A. Tramontani. Cutting Planes from the Branch-and-Bound Tree: Challenges and Opportunities. *INFORMS Journal on Computing*, 35(1):2–4, 2023.

[97] W. Cook. Fifty-plus years of combinatorial integer programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 387–430. Springer, Berlin, Heidelberg, 2010.

[98] W. Cook, S. Dash, R. Fukasawa, and M. Goycoolea. Numerically Safe Gomory Mixed-Integer Cuts. *INFORMS Journal on Computing*, 21(4):641–649, 2009.

[99] W. Cook, T. Koch, D. E. Steffy, and K. Wolter. An Exact Rational Mixed-Integer Programming Solver. In O. Günlük and G. J. Woeginger, editors, *Integer Programming and Combinatoral Optimization*, Lecture Notes in Computer Science, pages 104–116, Berlin, Heidelberg, 2011. Springer.

[100] W. Cook, T. Koch, D. E. Steffy, and K. Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5(3):305–344, 2013.

[101] W. J. Cook. *In pursuit of the traveling salesman*. Princeton University Press, Princeton, 2011.

[102] W. J. Cook, R. Kannan, and A. Schrijver. Chvátal closures for mixed integer programming problems. *Mathematical Programming*, 47:155–174, 1990.

[103] J. Cordeau, F. Furini, and I. Ljubić. Benders decomposition for very large scale partial set covering and maximal covering location problems. *European Journal of Operational Research*, 275(3):882–896, 2019.

[104] G. Cornuéjols. Revival of the Gomory cuts in the 1990's. *Annals of Operations Research*, 149(1):63–66, 2007.

[105] G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical programming*, 112(1):3–44, 2008.

[106] G. Cornuéjols and Y. Li. Elementary closures for integer programs. *Operations Research Letters*, 28(1):1–8, 2001.

[107] G. Cornuéjols, F. Margot, and G. Nannicini. On the safety of Gomory cut generators. *Mathematical Programming Computation*, 5(4):345–395, 2013.

[108] A. M. Costa. A survey on Benders decomposition applied to fixed-charge network design problems. *Computers & Operations Research*, 32(6):1429–1450, 2005.

[109] M.-C. Côté, B. Gendron, and L.-M. Rousseau. Grammar-Based Column Generation for Personalized Multi-Activity Shift Scheduling. *INFORMS Journal on Computing*, 25(3):461–474, 2013.

[110] T. G. Crainic, M. Hewitt, F. Maggioni, and W. Rei. Partial Benders decomposition: general methodology and application to stochastic network design. *Transportation Science*, 55(2):414–435, 2021.

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

[112] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[113] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1991.

[114] G. B. Dantzig and P. Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8(1):101–111, 1960.

[115] M. Daryalal, M. Bodur, and J. R. Luedtke. Lagrangian dual decision rules for multistage stochastic mixed-integer programming. *Operations Research*, 72(2):717–737, 2024.

[116] S. Dash and M. Goycoolea. A heuristic to generate rank-1 GMI cuts. *Mathematical Programming Computation*, 2(3):231–257, 2010.

[117] S. Dash, O. Günlük, and D. Lee. On a generalization of the Chvátal–Gomory closure. *Mathematical Programming*, 192(1-2):149–175, 2022.

[118] S. Dash, O. Günlük, and M. Molinaro. On the relative strength of different generalizations of split cuts. *Discrete Optimization*, 16:36–50, 2015.

[119] V. L. de Lima, C. Alves, F. Clautiaux, M. Iori, and J. M. V. de Carvalho. Arc Flow Formulations Based on Dynamic Programming: Theoretical Foundations and Applications. *European Journal of Operational Research*, 296(1):3–21, 2022.

[120] V. L. de Lima, M. Iori, and F. K. Miyazawa. New Exact Techniques Applied to a Class of Network Flow Formulations. In M. Singh and D. P. Williamson, editors, *Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science, pages 178–192, Cham, 2021. Springer International Publishing.

[121] M. Delorme and M. Iori. Enhanced Pseudo-polynomial Formulations for Bin Packing and Cutting Stock Problems. *INFORMS Journal on Computing*, 32(1):101–119, 2020.

[122] S. Demassey, G. Pesant, and L.-M. Rousseau. Constraint Programming Based Column Generation for Employee Timetabling. In R. Barták and M. Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 140–154, Berlin, Heidelberg, 2005. Springer.

[123] G. Desaulniers, F. Lessard, and A. Hadjar. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3):387–404, 2008.

[124] G. Desaulniers, F. Lessard, M. Saddoune, and F. Soumis. Dynamic Constraint Aggregation for Solving Very Large-scale Airline Crew Pairing Problems. *SN Operations Research Forum*, 1(3):19, 2020.

[125] J. Desrosiers, J. B. Gauthier, and M. E. Lübbecke. Row-reduced column generation for degenerate master problems. *European Journal of Operational Research*, 236(2):453–460, 2014.

[126] J. Desrosiers and M. E. Lübbecke. A Primer in Column Generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 1–32. Springer US, Boston, MA, 2005.

[127] S. S. Dey, Y. Dubey, M. Molinaro, and P. Shah. A theoretical and computational analysis of full strong-branching. *Mathematical Programming*, 205(1):303–336, 2024.

[128] S. S. Dey and M. Molinaro. Theoretical challenges towards cutting-plane selection. *Mathematical Programming*, 170(1):237–266, 2018.

[129] A. Deza and E. B. Khalil. Machine learning for cutting planes in integer programming: A survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI-2023. International Joint Conferences on Artificial Intelligence Organization, 2023.

[130] G. Di Liberto, S. Kadioglu, K. Leo, and Y. Malitsky. Dash: Dynamic approach for switching heuristics. *European Journal of Operational Research*, 248(3):943–953, 2016.

[131] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1):229–237, 1999.

[132] C. Duran-Mateluna, Z. Ales, and S. Elloumi. An efficient Benders decomposition for the p-median problem. *European Journal of Operational Research*, 2022.

[133] L. Eifler and A. Gleixner. A computational status update for exact rational mixed integer programming. *Mathematical Programming*, 197(2):793–812, 2023.

[134] L. Eifler and A. Gleixner. Safe and verified gomory mixed-integer cuts in a rational mixed-integer program framework. *SIAM Journal on Optimization*, 34(1):742–763, 2024.

[135] F. Eisenbrand. On the membership problem for the elementary closure of a polyhedron. *Combinatorica*, 19(2):297–300, 1999.

[136] I. Elhallaoui, A. Metrane, G. Desaulniers, and F. Soumis. An Improved Primal Simplex Algorithm for Degenerate Linear Programs. *INFORMS Journal on Computing*, 23(4):569–577, 2011.

[137] I. Elhallaoui, D. Villeneuve, F. Soumis, and G. Desaulniers. Dynamic Aggregation of Set-Partitioning Constraints in Column Generation. *Operations Research*, 53(4):632–645, 2005.

[138] A. S. Estes and M. O. Ball. Facets of the Stochastic Network Flow Problem. *SIAM Journal on Optimization*, 30(3):2355–2378, 2020.

[139] S. P. Fekete and J. Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, 91(1):11–31, 2001.

[140] FICO. FICO Xpress Optimization, 2024. Accessed: 2024-06-30.

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

[142] M. Fischetti, I. Ljubić, and M. Sinnl. Benders decomposition without separability: A computational study for capacitated facility location problems. *European Journal of Operational Research*, 253(3):557–569, 2016.

[143] M. Fischetti, I. Ljubić, and M. Sinnl. Redesigning Benders decomposition for large-scale facility location. *Management Science*, 63(7):2146–2162, 2017.

[144] M. Fischetti and A. Lodi. *Heuristics in Mixed Integer Programming*. John Wiley & Sons, Ltd.

[145] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1-3):23–47, 2003.

[146] M. Fischetti and A. Lodi. Optimizing over the first Chvátal closure. *Mathematical Programming*, 110(1):3–20, 2007.

[147] M. Fischetti, A. Lodi, M. Monaci, D. Salvagnin, and A. Tramontani. Tree Search Stabilization by Random Sampling. Technical report, 2013.

[148] M. Fischetti, A. Lodi, and A. Tramontani. On the separation of disjunctive cuts. *Mathematical Programming*, 128(1-2):205–230, 2011.

[149] M. Fischetti and M. Monaci. Backdoor Branching. In O. Günlük and G. J. Woeginger, editors, *Integer Programming and Combinatoral Optimization*, Lecture Notes in Computer Science, pages 183–191, Berlin, Heidelberg, 2011. Springer.

[150] M. Fischetti and M. Monaci. Branching on nonchimerical fractionalities. *Operations Research Letters*, 40(3):159–164, 2012.

[151] M. Fischetti and M. Monaci. Exploiting Erraticism in Search. *Operations Research*, 62(1):114–122, 2014.

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

[153] M. Fischetti and D. Salvagnin. A relax-and-cut framework for Gomory mixed-integer cuts. *Mathematical Programming Computation*, 3(2):79–102, 2011.

[154] M. Fischetti, D. Salvagnin, and A. Zanette. A note on the selection of Benders' cuts. *Mathematical Programming*, 124(1–2):175–182, 2010.

[155] N. Forget and S. N. Parragh. Enhancing branch-and-bound for multiobjective 0-1 programming. *INFORMS Journal on Computing*, 36(1):285–304, 2024.

[156] T. C.-O. Foundation. COIN-OR: Computational Infrastructure for Operations Research, 2024. Accessed: 2024-06-30.

[157] R. Fukasawa, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, and R. F. Werneck. Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem. In D. Bienstock and G. Nemhauser, editors, *Integer Programming and Combinatorial Optimization*, pages 1–15, Berlin, Heidelberg, 2004. Springer.

[158] R. Fukasawa and L. Poirrier. Numerically Safe Lower Bounds for the Capacitated Vehicle Routing Problem. *INFORMS Journal on Computing*, 29(3):544–557, 2017.

[159] E. Gaar and M. Sinnl. A scaleable projection-based branch-and-cut algorithm for the p-center problem. *European Journal of Operational Research*, 303(1):78–98, 2022.

[160] D. Gade, S. Küçükyavuz, and S. Sen. Decomposition algorithms with parametric Gomory cuts for two-stage stochastic integer programs. *Mathematical Programming*, 144(1-2):39–64, 2014.

[161] G. Gamrath, T. Koch, A. Martin, M. Miltenberger, and D. Weninger. Progress in presolving for mixed integer programming. *Mathematical Programming Computation*, 7(4):367–398, 2015.

[162] G. Gamrath and M. E. Lübbecke. Experiments with a Generic Dantzig-Wolfe Decomposition for Integer Programs. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and P. Festa, editors, *Experimental Algorithms*, volume 6049, pages 239–252. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[163] M. Gasse, D. Chetelat, N. Ferroni, L. Charlin, and A. Lodi. Exact Combinatorial Optimization with Graph Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[164] S. Gélinas, M. Desrochers, J. Desrosiers, and M. M. Solomon. A new branching strategy for time constrained routing problems with application to backhauling. *Annals of Operations Research*, 61(1):91–109, 1995.

[165] P. Gemander, W.-K. Chen, D. Weninger, L. Gottwald, A. Gleixner, and A. Martin. Two-row and two-column mixed-integer presolve using hashing-based pairing methods. *EURO Journal on Computational Optimization*, 8(3):205–240, 2020.

[166] A. Geoffrion. Generalized Benders Decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[167] P. C. Gilmore and R. E. Gomory. A Linear Programming Approach to the Cutting-Stock Problem. *Operations Research*, 9(6):849–859, 1961.

[168] W. Glankwamdee and J. Linderoth. Lookahead Branching for Mixed Integer Programming. In *12th INFORMS Computing Society Conference*, pages 130–148. INFORMS, 2011.

[169] J. Gleeson and J. Ryan. Identifying minimally infeasible subsystems of inequalities. *INFORMS Journal on Computing*, 2(1):61–63, 1990.

[170] A. Gleixner and D. E. Steffy. Linear programming using limited-precision oracles. *Mathematical Programming*, 183(1):525–554, 2020.

[171] F. Glover, G. Kochenberger, and Y. Du. A Tutorial on Formulating and Using QUBO Models. https://arxiv.org/abs/1811.11538, 2019.

[172] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[173] R. E. Gomory. An algorithm for the mixed integer problem. Technical Report Research Memorandum RM-2597, The Rand Corporation, 1960.

[174] J. Gondzio, P. González-Brevis, and P. Munari. New developments in the primal–dual column generation technique. *European Journal of Operational Research*, 224(1):41–51, 2013.

[175] Google. OR-Tools, 2024. Accessed: 2024-06-30.

[176] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[177] M. Grötschel and M. Padberg. Polyhedral theory. In E. Lawler, J. Lenstra, A. Rinooy Kan, and D. Shmoys, editors, *The Traveling Salesman Problem*, pages 251–306. John Wiley, 1985.

[178] T. Gschwind and S. Irnich. Stabilized column generation for the temporal knapsack problem using dual-optimal inequalities. *OR Spectrum*, 39(2):541–556, 2017.

[179] Gurobi. Gurobi Optimizer, 2024. Accessed: 2024-06-30.

[180] H. He, H. Daume III, and J. M. Eisner. Learning to Search in Branch and Bound Algorithms. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[181] E. Hellsten, D. F. Koza, I. Contreras, J.-F. Cordeau, and D. Pisinger. The transit time constrained fixed charge multi-commodity network design problem. *Computers & Operations Research*, 136:105511, 2021.

[182] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[183] C. Hojny. Packing, partitioning, and covering symresacks. *Discrete Applied Mathematics*, 283:689–717, 2020.

[184] C. Hojny and M. E. Pfetsch. Polytopes associated with symmetry handling. *Mathematical Programming*, 175(1-2):197–240, 2019.

[185] J. Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley, New York, 2000.

[186] J. Hooker. *Logic-Based Benders Decomposition: Theory and Applications*. Springer Cham, New York, 2023.

[187] J. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[188] J. N. Hooker. Planning and scheduling by logic-based Benders decomposition. *Operations Research*, 55(3):588–602, 2007.

[189] J. N. Hooker and W.-J. van Hoeve. Constraint programming and operations research. *Constraints*, 23(2):172–195, 2018.

[190] W. Huang, T. Huang, A. M. Ferber, and B. Dilkina. Distributional MIPLIB: a Multi-Domain Library for Advancing ML-Guided MILP Methods. `https://arxiv.org/abs/2406.06954`, 2024.

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

[192] J. Huchette, G. Muñoz, T. Serra, and C. Tsay. When deep learning meets polyhedral theory: A survey. `https://arxiv.org/abs/2305.00241`, 2023.

[193] T. Ibaraki. Successive sublimation methods for dynamic programming computation. *Annals of Operations Research*, 11(1):397–439, Dec. 1987.

[194] IBM. IBM ILOG CPLEX Optimizer, 2024. Accessed: 2024-06-30.

[195] V. Jain and I. E. Grossmann. Algorithms for Hybrid MILP/CP Models for a Class of Optimization Problems. *INFORMS Journal on Computing*, 13(4):258–276, 2001.

[196] M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors. *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. Springer, 2010.

[197] M. Jünger, E. Lobe, P. Mutzel, G. Reinelt, F. Rendl, G. Rinaldi, and T. Stollenwerk. Quantum annealing versus digital computing: An experimental comparison. *Journal of Experimental Algorithmics (JEA)*, 26:1–30, 2021.

[198] M. Jünger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

[199] U. Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, page 167–172. AAAI Press, 2004.

[200] V. Kaibel, M. Peinhardt, and M. E. Pfetsch. Orbitopal fixing. *Discrete Optimization*, 8(4):595–610, 2011.

[201] K. Kaparis and A. N. Letchford. Separation algorithms for 0-1 knapsack polytopes. *Mathematical Programming*, 124(1-2):69–91, 2010.

[202] E. Karlsson and E. Rönnberg. Logic-based Benders decomposition with a partial assignment acceleration technique for avionics scheduling. *Computers & Operations Research*, 146:105916, 2022.

[203] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[204] J. E. J. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[205] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[206] E. B. Khalil, B. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao. Learning to run heuristics in tree search. In *IJCAI*, pages 659–666, 2017.

[207] F. Kılınç Karzan, G. L. Nemhauser, and M. W. P. Savelsbergh. Information-based branching schemes for binary linear mixed integer problems. *Mathematical Programming Computation*, 1(4):249–293, 2009.

[208] T. Kleinert, M. Labbé, I. Ljubić, and M. Schmidt. A survey on mixed-integer programming techniques in bilevel optimization. *EURO Journal of Computational Optimization*, 9:100007, 2021.

[209] A. J. Kleywegt, A. Shapiro, and T. Homem-de-Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002.

[210] T. Koch, T. Berthold, J. Pedersen, and C. Vanaret. Progress in mathematical programming solvers from 2001 to 2020. *EURO Journal on Computational Optimization*, 10:100031, 2022.

[211] M. Köppe, C. T. Ryan, and M. Queyranne. Rational generating functions and integer programming games. *Operations Research*, 59(6):1445–1460, 2011.

[212] M. Köppe and J. Wang. Dual-feasible functions for integer programming and combinatorial optimization: Algorithms, characterizations, and approximations. *Discrete Applied Mathematics*, 308:84–106, 2022.

[213] A. Kramer, M. Dell'Amico, and M. Iori. Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines. *European Journal of Operational Research*, 275(1):67–79, 2019.

[214] S. Kraul, M. Seizinger, and J. O. Brunner. Machine Learning–Supported Prediction of Dual Variables for the Cutting Stock Problem with an Application in Stabilized Column Generation. *INFORMS Journal on Computing*, 35(3):692–709, 2023.

[215] M. Kruber, M. E. Lübbecke, and A. Parmentier. Learning When to Use a Decomposition. In D. Salvagnin and M. Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, Lecture Notes in Computer Science, pages 202–210, Cham, 2017. Springer International Publishing.

[216] Y. Kurokawa, R. Mitsuboshi, H. Hamasaki, K. Hatano, E. Takimoto, and H. Rahmanian. Extended formulations via decision diagrams. In *International Computing and Combinatorics Conference*, pages 17–28. Springer, 2023.

[217] J. Kurtz, Ş. İ. Birbil, and D. den Hertog. Counterfactual Explanations for Linear Optimization. `https://arxiv.org/abs/2405.15431`, 2024.

[218] A. G. Labassi, D. Chételat, and A. Lodi. Learning to compare nodes in branch and bound with graph neural networks. *Advances in neural information processing systems*, 35:32000–32010, 2022.

[219] E. Lam, G. Gange, P. J. Stuckey, P. V. Hentenryck, and J. J. Dekker. Nutmeg: a MIP and CP hybrid solver using branch-and-check. *Operations Research Forum*, 1(3), 2020.

[220] S. Lamontagne, M. Carvalho, and R. Atallah. Accelerated Benders decomposition and local branching for dynamic maximum covering location problems. *Computers & Operations Research*, 167:106673, 2024.

[221] A. Land and A. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[222] G. Laporte and F. V. Louveaux. The integer l-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, 13(3):133–142, 1993.

[223] P. Le Bodic and G. Nemhauser. An abstract model for branching and its application to mixed integer programming. *Mathematical Programming*, 166(1):369–405, 2017.

[224] M. Leitner, I. Ljubić, M. Luipersbeck, and M. Sinnl. Decomposition methods for the two-stage stochastic Steiner tree problem. *Computational Optimization and Applications*, 69(3):713–752, 2018.

[225] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov. New variants of bundle methods. *Mathematical Programming*, 69(1-3):111–147, 1995.

[226] M. Lerouge, C. Gicquel, V. Mousseau, and W. Ouerdane. Counterfactual Explanations for Workforce Scheduling and Routing Problems. In *12th International Conference on Operations Research and Enterprise Systems*, pages 50–61, 2024.

[227] A. N. Letchford and G. Souli. On lifted cover inequalities: A new lifting procedure with unusual properties. *Operations Research Letters*, 47(2):83–87, 2019.

[228] S. Li, W. Ouyang, M. Paulus, and C. Wu. Learning to Configure Separators in Branch-and-Cut. *Advances in Neural Information Processing Systems*, 36:60021–60034, 2023.

[229] L. Liberti. Reformulations in mathematical programming: Automatic symmetry detection and exploitation. *Mathematical Programming*, 131(1):273–304, 2012.

[230] L. Liberti and J. Ostrowski. Stabilizer-based symmetry breaking constraints for mathematical programs. *Journal of Global Optimization*, 60(2):183–194, 2014.

[231] J. Linderoth, J. Núñez Ares, J. Ostrowski, F. Rossi, and S. Smriglio. Orbital Conflict: Cutting Planes for Symmetric Integer Programs. *INFORMS Journal on Optimization*, 3(2):139–153, 2021.

[232] J. T. Linderoth and M. W. P. Savelsbergh. A Computational Study of Search Strategies for Mixed Integer Programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[233] I. Ljubić, P. Mutzel, and B. Zey. Stochastic survivable network design problems: Theory and practice. *European Journal of Operational Research*, 256(2):333–348, 2017.

[234] I. Ljubić, M. A. Pozo, J. Puerto, and A. Torrejón. Benders decomposition for the discrete ordered median problem. *European Journal of Operational Research*, 317(3):858–874, 2024.

[235] A. Lodi. *Mixed Integer Programming Computation*, pages 619–645. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[236] A. Lodi and A. Tramontani. Performance Variability in Mixed-Integer Programming. In *Theory Driven by Influential Applications*, INFORMS TutORials in Operations Research, chapter 1, pages 1–12. INFORMS, 2013.

[237] A. Lodi and G. Zarpellon. On learning and branching: A survey. *TOP*, 25(2):207–236, 2017.

[238] H. Lu. First-order methods for linear programming. `https://arxiv.org/abs/2403.14535`, 2024.

[239] M. Lübbecke and C. Puchert. Primal Heuristics for Branch-and-Price Algorithms. In D. Klatte, H.-J. Lüthi, and K. Schmedders, editors, *Operations Research Proceedings 2011*, Operations Research Proceedings, pages 65–70, Berlin, Heidelberg, 2012. Springer.

[240] B. Luteberget and G. Sartor. Feasibility Jump: an LP-free Lagrangian MIP heuristic. *Mathematical Programming Compututation*, 15(2):365–388, 2023.

[241] T. L. Magnanti and R. T. Wong. Accelerating Benders decomposition: Algorithmic enhancement and model selection criteria. *Operations Research*, 29(3):464–484, 1981.

[242] S. J. Maher. Implementing the branch-and-cut approach for a general purpose Benders' decomposition framework. *European Journal of Operational Research*, 290(2):479–498, 2021.

[243] D. Maragno, J. Kurtz, T. E. Röber, R. Goedhart, Ş. I. Birbil, and D. den Hertog. Finding regions of counterfactual explanations via robust optimization. *INFORMS Journal on Computing*, 2025.

[244] H. Marchand, A. Martin, R. Weismantel, and L. Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1-3):397–446, 2002.

[245] F. Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94(1):71–90, 2002.

[246] F. Margot. Symmetry in Integer Linear Programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 647–686. Springer, Berlin, Heidelberg, 2010.

[247] R. E. Marsten, W. W. Hogan, and J. W. Blankenship. The Boxstep Method for Large-Scale Optimization. *Operations Research*, 23(3):389–405, 1975.

[248] R. K. Martin, R. L. Rardin, and B. A. Campbell. Polyhedral Characterization of Discrete Dynamic Programming. *Operations Research*, 38(1):127–138, 1990.

[249] J. Martinovic, N. Strasdat, J. Valério de Carvalho, and F. Furini. A combinatorial flow-based formulation for temporal bin packing problems. *European Journal of Operational Research*, 307(2):554–574, 2023.

[250] C. C. McGeoch. Theory versus practice in annealing-based quantum computing. *Theoretical Computer Science*, 816:169–183, 2020.

[251] T. Mhamedi, H. Andersson, M. Cherkesly, and G. Desaulniers. A Branch-Price-and-Cut Algorithm for the Two-Echelon Vehicle Routing Problem with Time Windows. *Transportation Science*, 56(1):245–264, 2022.

[252] M. Morabit, G. Desaulniers, and A. Lodi. Machine-learning–based column selection for column generation. *Transportation Science*, 55(4):815–831, 2021.

[253] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[254] N. T. Moungla, L. Létocart, and A. Nagih. Solutions diversification in a column generation algorithm. *Algorithmic Operations Research*, 5(2):86–95, 2010.

[255] P. Munari and J. Gondzio. Using the primal-dual interior point algorithm within the branch-price-and-cut method. *Computers & Operations Research*, 40(8):2026–2036, 2013.

[256] G. Muñoz, D. Espinoza, M. Goycoolea, E. Moreno, M. Queyranne, and O. R. Letelier. A study of the Bienstock–Zuckerberg algorithm: Applications in mining and resource constrained project scheduling. *Computational Optimization and Applications*, 69(2):501–534, 2018.

[257] İ. Muter, Ş. İ. Birbil, and K. Bülbül. Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows. *Mathematical Programming*, 142(1):47–82, 2013.

[258] D. Naddef. Polyhedral theory and branch-and-cut algorithms for the symmetric tsp. In G. Gutin and A. P. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, pages 29–116. Springer US, Boston, MA, 2007.

[259] J. Naoum-Sawaya. Recursive central rounding for mixed integer programs. *Computers & Operations Research*, 43:191–200, 2014.

[260] J. Naoum-Sawaya and S. Elhedhli. An interior-point Benders based branch-and-cut algorithm for mixed integer programs. *Annals of Operations Research*, 210(1):33–55, 2013.

[261] G. L. Nemhauser and L. A. Wolsey. A recursive procedure to generate all cuts for 0–1 mixed integer programs. *Mathematical Programming*, 46(1):379–390, 1990.

[262] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio. Orbital branching. *Mathematical Programming*, 126(1):147–178, 2011.

[263] M. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope. *Mathematical Programming*, 47:219–257, 1990.

[264] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[265] M. Padberg and T. Sung. An analytical comparison of different formulations of the travelling salesman problem. *Mathematical Programming*, 52:315–357, 1991.

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

[267] N. Papadakos. Practical enhancements to the Magnanti–Wong method. *Operations Research Letters*, 36(4):444–449, 2008.

[268] M. B. Paulus, G. Zarpellon, A. Krause, L. Charlin, and C. J. Maddison. Learning To Cut By Looking Ahead: Imitation Learning for Cutting Plane Selection. In Chaudhuri, Kamalika, Jegelka, Stefanie, Song, Le, Szepesvari, Csaba, Niu, Gang, and Sabato, Sivan, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 17584–17600. PMLR, 2022.

[269] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, 2017.

[270] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. Automation and Combination of Linear-Programming Based Stabilization Techniques in Column Generation. *INFORMS Journal on Computing*, 30(2):339–360, 2018.

[271] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183(1):483–523, 2020.

[272] M. E. Pfetsch and T. Rehn. A computational comparison of symmetry handling methods for mixed integer programs. *Mathematical Programming Computation*, 11(1):37–93, 2019.

[273] T. Polzin and S. Vahdati Daneshmand. A comparison of Steiner tree relaxations. *Discrete Applied Mathematics*, 112(1):241–261, 2001.

[274] D. Porumbel and F. Clautiaux. Constraint aggregation in column generation models for resource-constrained covering problems. *INFORMS J. Comput.*, 29(1):170–184, 2017.

[275] D. C. Porumbel and G. Goncalves. Using dual feasible functions to construct fast lower bounds for routing and location problems. *Discrete Applied Mathematics*, 196:83–99, 2015.

[276] R. Rahmaniani, S. Ahmed, T. G. Crainic, M. Gendreau, and W. Rei. The Benders dual decomposition method. *Operations Research*, 68(3):878–895, 2020.

[277] R. Rahmaniani, T. G. Crainic, M. Gendreau, and W. Rei. The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017.

[278] R. Rahmaniani, T. G. Crainic, M. Gendreau, and W. Rei. Accelerating the Benders decomposition method: Application to stochastic network design problems. *SIAM Journal on Optimization*, 28(1):875–903, 2018.

[279] T. Ralphs, Y. Shinano, T. Berthold, and T. Koch. Parallel solvers for mixed integer linear programming. Technical report, Zuse Institute Berlin, 2016.

[280] C. Ramirez-Pico, I. Ljubić, and E. Moreno. Benders adaptive-cuts method for two-stage stochastic programs. *Transportation Science*, 57(5):1252–1275, 2023.

[281] C. Ramirez-Pico and E. Moreno. Generalized adaptive partition-based method for two-stage stochastic linear programs with fixed recourse. *Mathematical Programming*, 196:755–774, 2022.

[282] J. Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical programming*, 40(1):59–93, 1988.

[283] G. Righini and M. Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51:155 – 170, 05 2008.

[284] E. Rothberg. An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

[285] L.-M. Rousseau, M. Gendreau, and D. Feillet. Interior point stabilization for column generation. *Operations Research Letters*, 35(5):660–668, 2007.

[286] D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, pages 269–280. North-Holland Publishing Company, Amsterdam, 1981.

[287] A. Sabharwal, H. Samulowitz, and C. Reddy. Guiding Combinatorial Optimization with UCT. In N. Beldiceanu, N. Jussien, and É. Pinson, editors, *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 356–361, Berlin, Heidelberg, 2012. Springer.

[288] R. Sadykov. *Modern Branch-Cut-and-Price*. PhD thesis, Université de Bordeaux, 2019.

[289] R. Sadykov, E. Uchoa, and A. Pessoa. A Bucket Graph–Based Labeling Algorithm with Application to Vehicle Routing. *Transportation Science*, 55(1):4–28, 2021.

[290] R. Sadykov and F. Vanderbeck. Column generation for extended formulations. *EURO Journal on Computational Optimization*, 1(1):81–115, 2013.

[291] R. Sadykov, F. Vanderbeck, A. Pessoa, I. Tahiri, and E. Uchoa. Primal Heuristics for Branch and Price: The Assets of Diving Methods. *INFORMS Journal on Computing*, 31(2):251–267, 2019.

[292] A. Saken, E. Karlsson, S. J. Maher, and E. Rönnberg. Computational evaluation of cut-strengthening techniques in logic-based Benders' decomposition. *Operations Research Forum*, 4(3):62, 2023.

[293] H. Salemi and D. Davarnia. On the Structure of Decision Diagram–Representable Mixed-Integer Programs with Application to Unit Commitment. *Operations Research*, page opre.2022.2353, 2022.

[294] A. Schrijver. *Combinatorial Optimization*. Springer, 2002.

[295] SCIP. SCIP Optimization Suite, 2024. Accessed: 2024-06-30.

[296] K. Seo, S. Joung, C. Lee, and S. Park. A closest Benders cut selection scheme for accelerating the Benders decomposition algorithm. *INFORMS Journal on Computing*, 34(5):2804–2827, 2022.

[297] Y. Shen, Y. Sun, A. Eberhard, and X. Li. Learning primal heuristics for mixed integer programs. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[298] R. S. Shibasaki, M. Baiou, F. Barahona, P. Mahey, and M. C. de Souza. Lagrangian bounds for large-scale multicommodity network design: A comparison between Volume and Bundle methods. *International Transactions in Operational Research*, 28(1):296–326, 2021.

[299] Y. Shinano, S. Heinz, S. Vigerske, and M. Winkler. FiberSCIP—A Shared Memory Parallelization of SCIP. *INFORMS Journal on Computing*, 30(1):11–30, 2018.

[300] G. Song and R. Leus. Parallel machine scheduling under uncertainty: Models and exact algorithms. *INFORMS Journal on Computing*, 34(6):3059–3079, 2022.

[301] J. Song, r. lanka, Y. Yue, and B. Dilkina. A general large neighborhood search framework for solving integer linear programs. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20012–20023. Curran Associates, Inc., 2020.

[302] J. Song, R. Lanka, A. Zhao, A. Bhatnagar, Y. Yue, and M. Ono. Learning to Search via Retrospective Imitation. `https://arxiv.org/abs/1804.00846`, 2019.

[303] J. Song, Y. Yue, B. Dilkina, et al. A general large neighborhood search framework for solving integer linear programs. *Advances in Neural Information Processing Systems*, 33:20012–20023, 2020.

[304] Y. Song and J. Luedtke. An adaptive partition-based approach for solving two-stage stochastic programs with fixed recourse. *SIAM Journal on Optimization*, 25(3):1344–1367, 2015.

[305] S. Tahernejad, T. K. Ralphs, and S. T. DeNegre. A branch-and-cut algorithm for mixed integer bilevel linear optimization problems and its implementation. *Mathematical Programming Computation*, 12(4):529–568, 2020.

[306] Y. Tang, S. Agrawal, and Y. Faenza. Reinforcement Learning for Integer Programming: Learning to Cut. In *Proceedings of the 37th International Conference on Machine Learning*, pages 9367–9376. PMLR, 2020.

[307] D. Thuerck, B. Sofranac, M. E. Pfetsch, and S. Pokutta. Learning cuts via enumeration oracles. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 79108–79123. Curran Associates, Inc., 2023.

[308] S. Trukhanov, L. Ntaimo, and A. Schaefer. Adaptive multicut aggregation for two-stage stochastic linear programs with recourse. *European Journal of Operational Research*, 206(2):395–406, 2010.

[309] M. Turner, T. Berthold, M. Besançon, and T. Koch. Cutting Plane Selection with Analytic Centers and Multiregression. In A. A. Cire, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 52–68, Cham, 2023. Springer Nature Switzerland.

[310] E. Uchoa and R. Sadykov. Kantorovich and Zalgaller (1951): The 0-th Column Generation Algorithm. `https://optimization-online.org/2024/01/kantorovich-and-zalgaller-1951-the-0-th-column-generation-algorithm/`, 2024.

[311] J. M. Valério de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273, 2002.

[312] R. M. Van Slyke and R. Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4):638–663, 1969.

[313] F. Vanderbeck. On Dantzig-Wolfe Decomposition in Integer Programming and Ways to Perform Branching in a Branch-and-Price Algorithm. *Operations Research*, 48(1):111, 2000.

[314] F. Vanderbeck. Branching in branch-and-price: A generic scheme. *Mathematical Programming*, 130(2):249–294, 2011.

[315] F. Vanderbeck and M. W. P. Savelsbergh. A generic view of Dantzig–Wolfe decomposition in mixed integer programming. *Operations Research Letters*, 34(3):296–306, 2006.

[316] F. Vanderbeck and L. A. Wolsey. Reformulation and Decomposition of Integer Programs. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 431–502. Springer, Berlin, Heidelberg, 2010.

[317] J. Witzig and A. Gleixner. Conflict-driven heuristics for mixed integer programming. *INFORMS Journal on Computing*, 33(2):706–720, 2021.

[318] L. A. Wolsey and G. L. Nemhauser. *Integer and combinatorial optimization*. John Wiley & Sons, 2014.

[319] Y. Yang. DeLuxing: Deep Lagrangian Underestimate Fixing for Column-Generation-Based Exact Methods. *Optimization Online*, (24217), 2023.

[320] Y. Yang, N. Boland, B. Dilkina, and M. Savelsbergh. Learning generalized strong branching for set covering, set packing, and 0–1 knapsack problems. *European Journal of Operational Research*, 301(3):828–840, 2022.

[321] K. Yilmaz and N. Yorke-Smith. A Study of Learning Search Approximation in Mixed Integer Branch and Bound: Node Selection in SCIP. *AI*, 2(2):150–178, 2021.

[322] A. Zaghrouti, I. El Hallaoui, and F. Soumis. Improved integral simplex using decomposition for the set partitioning problem. *EURO Journal on Computational Optimization*, 6(2):185–206, 2018.

[323] G. Zakeri, A. B. Philpott, and D. M. Ryan. Inexact cuts in Benders decomposition. *SIAM Journal on Optimization*, 10(3):643–657, 2000.

[324] G. Zarpellon, J. Jo, A. Lodi, and Y. Bengio. Parameterizing branch-and-bound search trees to learn branching policies. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 3931–3939, 2021.

[325] C. A. Zetina, I. Contreras, and J.-F. Cordeau. Exact algorithms based on Benders decomposition for multicommodity uncapacitated fixed-charge network design. *Computers & Operations Research*, 111:311–324, 2019.

[326] M. Zhang and S. Küçükyavuz. Finitely convergent decomposition algorithms for two-stage stochastic pure integer programs. *SIAM Journal of Optimization*, 24(4):1933–1951, 2014.