Neurocomputing 519 (2023) 205–217

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Survey paper

A survey for solving mixed integer programming via machine learning

Jiayi Zhang[a], Chang Liu[a], Xijun Li[b,*], Hui-Ling Zhen[b], Mingxuan Yuan[b], Yawen Li[c,*], Junchi Yan[a,*]

[a] Department of Computer Science and Engineering, and MoE Key Lab of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China
[b] Noah's Ark Lab, Huawei Ltd., Shenzhen, China
[c] School of Economics and Management, Beijing University of Posts and Telecommunications, Beijing, China

ARTICLE INFO

Article history:
Received 30 July 2022
Revised 12 October 2022
Accepted 4 November 2022
Available online 16 November 2022

Keywords:
Mixed integer programming
Machine learning
Combinatorial optimization

ABSTRACT

Machine learning (ML) has been recently introduced to solving optimization problems, especially for combinatorial optimization (CO) tasks. In this paper, we survey the trend of leveraging ML to solve the mixed-integer programming problem (MIP). Theoretically, MIP is an NP-hard problem, and most CO problems can be formulated as MIP. Like other CO problems, the human-designed heuristic algorithms for MIP rely on good initial solutions and cost a lot of computational resources. Therefore, researchers consider applying machine learning methods to solve MIP since ML-enhanced approaches can provide the solution based on the typical patterns from the training data. Specifically, we first introduce the formulation and preliminaries of MIP and representative traditional solvers. Then, we show the integration of machine learning and MIP with detailed discussions on related learning-based methods, which can be further classified into exact and heuristic algorithms. Finally, we propose the outlook for learning-based MIP solvers, the direction toward more combinatorial optimization problems beyond MIP, and the mutual embrace of traditional solvers and ML components. We maintain a list of papers that utilize machine learning technologies to solve combinatorial optimization problems, which is available at https://github.com/Thinklab-SJTU/awesome-ml4co.

1. Introduction

Mixed-integer programming problems (MIP) are significant parts of combinatorial optimization (CO) problems. Benefiting from academic theory and commercial software development, MIP has become a vital capability that powers a wide range of applications, including planning [1,2], scheduling [3,4], routing [5,6] and bin packing [7], etc.

One MIP problem may contain both integer and continuous variables. If the problem contains an objective function with only linear forms, then the problem is termed a Mixed Integer Linear Programming (MILP). Formally, it is given as:

$$\begin{aligned} &\min \mathbf{c}^\top \mathbf{x} \\ \text{s.t.}\, &\mathbf{A}\mathbf{x} \geqslant \mathbf{b}, \quad \mathbf{x} \in \mathbb{R}^n \quad x_j \in \mathbb{Z}, \forall j \in I \end{aligned} \tag{1}$$

When there is at least a quadratic term in the objective, the problem turns to a Mixed Integer Quadratic Program (MIQP); if the model has any constraint containing a quadratic term, regardless of the objective function, the problem is termed as a Mixed Integer Quadratic Constrained Program (MIQCP). In this paper, we only focus on the basic MILP, and we interchangeably use the term MIP and MILP throughout the paper.

The challenge in solving the MIP problem is that its feasible region is discrete and non-convex, which makes the feasible region difficult to analyze and optimization methods hard to design. Typically, we do not solve the MIP directly, but first, solve the corresponding LP relaxation and then round to obtain a near-optimal solution. The LP relaxation of MILP is obtained by omitting the integer requirements:

$$\begin{aligned} &\min \mathbf{c}^\top \mathbf{x} \\ \text{s.t.}\quad &\mathbf{A}\mathbf{x} \geqslant \mathbf{b}, \mathbf{x} \in \mathbb{R}^n \end{aligned} \tag{2}$$

LP relaxation can provide a (lower) bound for optimum MIP (for minimization problems), and the optimal solution of linear relaxation is often better than that of its raw form. The problem is then often solved by trying to increase the lower bound and decrease the upper bound to narrow down the range estimation of the optimal solution.

In general, solving the MIP is NP-hard. Many techniques have been designed to find high-quality solutions in a limited time.

* Corresponding authors.
E-mail addresses: zhangjiayirr@sjtu.edu.cn (J. Zhang), only-changer@sjtu.edu.cn (C. Liu), xijun.li@huawei.com (X. Li), zhenhuiling2@huawei.com (H.-L. Zhen), yuan.mingxuan@huawei.com (M. Yuan), warmly0716@126.com (Y. Li), yanjunchi@sjtu.edu.cn (J. Yan).

https://doi.org/10.1016/j.neucom.2022.11.024
0925-2312/© 2022 Elsevier B.V. All rights reserved.

Machine learning methods have shown success evidence solving some NP-hard combinatorial optimization problems[1] There is potential to assist MIP solvers with machine learning [8], in a way of better learning the problem solving heuristics in a data-driven manner, or in a mixed way for infusing algorithm design expertise as well.

There are emerging surveys in machine learning for combinatorial optimization [8] including those on some specific problems e.g. graph matching [9]. There are also excellent studies on learning for solving mixed-integer programming [10,11]. Specifically, the survey [10] mainly covers learning-supported node selection techniques due to the limited development in this area five years ago. A recent study [11] focuses on the development in recent decades of the branch-and-bound algorithm itself (with machine learning as well) while less tailored to the MIP problem, and it does not emphasize the importance of adapting machine learning in MIP. In this paper, we make efforts to give a comprehensive and up-to-date review in the area of machine learning for solving MIP, especially seeing the rapid development in recent years. Not only do we introduce methods based on the much-discussed branch-and-bound algorithm, but we also review several methods based on optimization heuristics and solvers, showing a variety of different ideas.

In this paper, as shown in Fig. 1, we aim to organize and analyze the existing solvers to MIP in two main threads. One is the traditional approach based MIP solvers which also lay some foundation for the subsequent machine learning techniques adopted in MIP. The other one is machine learning-based methods. Specifically, it can be further divided into two cases: technologies based on exact algorithms and those based on heuristic schemes for approximate solving.

2. Traditional approaches

In this section, we introduce several widely used traditional algorithms, which are divided into exact algorithms in Section 2.1 and heuristic algorithms in Section 2.2. These algorithms serve as the basis for machine learning models, and their introduction lays the groundwork for a better understanding of the machine learning models introduced later.

2.1. *Exact algorithms*

Branch-and-cut and decomposition are two widely used algorithms for exact solving mixed-integer linear programming problems. We introduce branch-and-cut in Section 2.1.1 and decomposition in Section 2.1.2 respectively, including the algorithm and the combination with machine learning.

2.1.1. Branch-and-cut

Branch and bound is a widely used technique for exactly solving MIP instances [12], which can serve as an exact algorithm proven to reach the optimal solution eventually [13]. Moreover, the branch-and-cut algorithm is currently the most popular architecture for MIP solvers, which is formed by adding the cutting plane on the basis of branch and bound. The cutting plane is used to tighten the feasible region of the optimal solution, thereby significantly speeding up the solving process. The branch-and-cut spends more time on solving the LP relaxation problem than the branch-and-bound, but the improved LP bounds by the cutting plane lead to a smaller search tree.

In this subsection, we first overview the general branch-and-bound algorithm and the two most crucial branching algorithms in branch-and-bound, including branching variable selection and node selection. Then, we introduce the cutting plane, which is part of the branch-and-cut method.

Overview of Branch-and-Bound: For minimization problems, the core idea of branch-and-bound is to decompose a MILP problem into solving LP problems (P problems) and record the upper bound (the most optimal feasible solution) and lower bound (optimal linear relaxation solution) of the original problem during the solution process. Branch-and-bound is a classic tree search method. By divide-and-conquer, it partitions the search space by branching on the value of and smartly uses bounds from problem relaxations to prune unpromising regions from the tree.

Define an optimization problem as $\mathscr{P} = (X, f)$, where X (called the search space) is a set of valid solutions, and $f : X \to \mathbb{R}$ is the objective function. The goal is to find an optimal solution $x^* \in \arg\min_{x \in X} f(x)$. In order to solve $\mathscr{P}$, a search tree T of subsets of the search space is built. Additionally, a feasible solution $\hat{x} \in X$ (called the incumbent solution) is stored globally. At each iteration, the algorithm selects a new subset $S \subseteq X$ to explore from a list L of unexplored subsets; if a solution $\hat{x}' \in S$ (called a candidate incumbent) can be found with a better objective value than $\hat{x}$ (i.e., $f(\hat{x}') < f(\hat{x})$), the incumbent solution is updated. On the other hand, if it can be proved that no solution in S has a better objective value than $\hat{x}$(i.e. , $\forall x \in S, f(x) \geqslant f(\hat{x})$), the subset is pruned (or fathomed), and the subset is terminal. Otherwise, child subsets are generated by partitioning S into an exhaustive (but not necessarily mutually exclusive) set of subset $S_1, S_2, \ldots, S_r$, which are then inserted into T. Once no unexplored subset remain, the best incumbent solution is returned; since the subset is only fathomed if they contain no solution better than $\hat{x}$, it must be the case that $\hat{x} \in \arg\min_{x \in X} f(x)$. Pseudocode for the generic B&B procedure is given in Alg. 1, and an example of solving MIP by B&B [14] is shown in Fig. 2.

Algorithm 1: Branch-and-Bound (X, f) [15]

```
1  Set $L = \{X\}$ and
2  while $L \neq \emptyset$ do
3      Select a subproblem $S$ from $L$ to explore
4      if a solution $\hat{x}' \in \{x \in S \mid f(x) < f(\hat{x})\}$ can be found
         then
5          Set $\hat{x} = \hat{x}'$
6      end
7      if $S$ cannot be pruned then
8          Partition $S$ into $S_1, S_2, \ldots, S_r$
9          Insert $S_1, S_2, \ldots, S_r$ into $L$
10     end
11     Remove $S$ from $L$
12 end
   Output: $\hat{x}$
```

Branching Variable Selection (BVS): Branch variable selection determines which fractional variables (also known as candidates) to branch the current node into two child nodes. A score is used to measure the variable's effectiveness and pick the candidate with the highest score to branch on. The pseudocode of BVS is presented in Alg. 2, illustrating the basic variable selection idea.

[1] See our GitHub repository Awsome-ML4CO: https://github.com/Thinklab-SJTU/awesome-ml4co, which has received more than 700 stars. So far we cover up to 24 combinatorial optimization problems, including MIP.

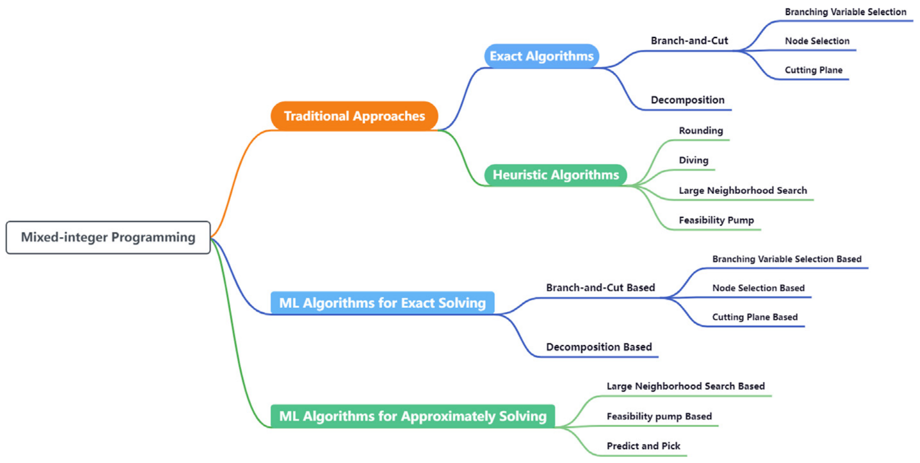


Fig. 1. Existing approaches overview and the organization of the survey. We discuss traditional non-learning methods in Section 2, followed by the two lines of learning-aided methods for exact and approximate solving in Section 3 and Section 4 respectively.

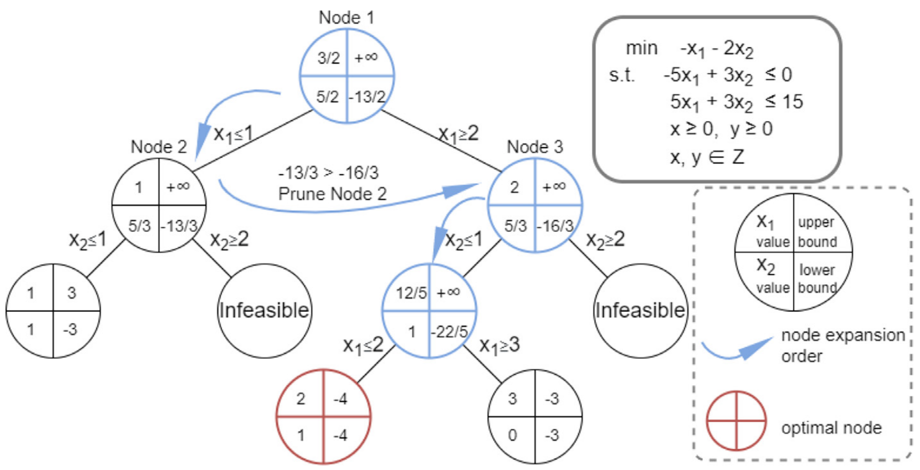


Fig. 2. Use branch-and-bound to solve an integer linear programming minimization. (Credit to Fig.1 of [14]).

Algorithm 2: Branching Variable Selection in B&B (credit to Algorithm 2 of [11])

```
Input: Subproblem of the current node $\mathcal{S}$ with its
       optimal LP solution $\hat{x} \notin X_{MILP}$
1 Define branching candidates set $C = \{i \in I \mid \hat{x}_i \notin \mathbb{Z}\}$
2 for each candidate $i \in C$ do
3     Calculate its score value $s_i \in \mathbb{R}$
4 end
Output: A subscript $i = \arg\min_{i \in C} s_i$ of an integer
        variable with fractional value $\hat{x}_i \notin \mathbb{Z}$
```

Various branching rules are used to measure variables in BVS, including strong branching (SB) rule [16], pseudo-cost branching [17] and hybrid branching [18].

Strong branching (SB) keeps the B&B tree as small as possible. It tentatively computes linear relaxation, gets potential lower bound improvement on candidate branch variables before actual branching is performed, and selects the variable with the most remarkable improvement of bound to branch. It becomes full strong branching (FSB) if all branching variables are computed. However, the enormous computational overhead issue in the strong branching strategy is unacceptable in practice.

Pseudo-cost branching keeps track of success variables already branched on, predicts the impact of candidate branch variables on the value of the objective function, and selects the variable that has the tremendous change on the value of the objective function to branch. While pseudo-cost branching has high computational time efficiency, the branching lacks reliable history at the beginning of a B&B process. Although pseudo-cost branching is very efficient in computation time, the branching performed at the beginning of the B&B tree might be inefficient as no reliable history has been recorded at that time. Moreover, hybrid branching performs strong branching at the beginning of the solution process and then switches to other strategies like pseudo-cost branching.

Node Selection: As mentioned above, the branch-and-bound algorithm recursively divides the feasible set of a problem into disjoint subsets, organized in a tree structure, where each node represents a subproblem that searches only the subset at that node. The main steps of the node selection algorithm are given in Alg. 3. If computing bounds on a subproblem does not rule out the possibility that its subset contains the optimal solution, the subset can be further partitioned ("branched ") as needed. Else if the lower bound of possible solutions of a node is larger than the known upper bound, the node can be pruned. A key question in B&B is how to prioritize which nodes to be considered. An effective node priority decision strategy guides the tree search to promising areas and improves the chance of quickly finding an excellent incumbent solution, which can be used to decide whether to discard or expand other nodes. Thus, learning the appropriate node selection policy of a B&B tree is worthy of being investigated. More precisely, a policy is a function that maps a state to an action, which in this case is the next node to be selected.

Algorithm 3: Node Selection in B&B

```
   Input: Node list $L = \{P\}$ and upper bound $\bar{J}$
1  Assigns a score to each active node
2  Pop a node $P^i$ from $L$
3  Solve the LP relaxation
4  get solution$(x^i, y^i)$and lower bound $J^i$
5  if $J^i \geq \bar{J}$ then
6      Prune the node $P^i$
7  end
8  else if $y^i \in \mathbb{Z}^p$ then
9      Improve upper bound $\bar{J} = J^i$
10 end
11 else
12     Branch node $P^i$ and push child nodes to list $L$
13 end
```

There are various node selection rules concerning which node (and hence which LP) to solve next: 1) Breath-first search explores the node branch as far as possible before backtracking and expanding other nodes; 2) Depth-first search starts at the root node and explores all nodes at the current depth before moving to nodes at the next depth level. It has a lower memory requirement and usually finds a feasible solution quickly than the breath-first search, but it may spend a lot of time on a bad branch.

Cutting Plane: It is known that every MIP can be relaxed to linear programming (LP) by dropping the integer constraints, and there are many traditional efficient algorithms for solving LP, such as Simplex [19] and Interior Point Method (IPM) [20]. It means we can relax MIP to LP and solve the incident LP at an acceptable cost, leading to the cutting plane algorithm.

Cutting plane methods add cuts to the LP relaxation iteratively. The cuts are linear constraints that can tighten the LP relaxation by eliminating some part of the feasible region while preserving the LP optimal solution. As Fig. 3 shows, the area enclosed by the blue line represents the LP relaxation, and the solution x^r is infeasible. Then, cut1 (green line) is added to the LP relaxation, separating the infeasible point x^r and tights the LP relaxation by cutting some part of the feasible region. However, the LP solution x' is still infeasible, and cut2 (orange line) is added to LP relaxation. Finally, the optimal solution x'' is found. Suppose that we add the cut set (valid inequalities(α_i, β_i)) $C' = \{\alpha_i^\top x \geqslant \beta_i\}_{i=1}^{|C'|}$to the original formulation in Eq. (1). Then, the optimization formulation of MIP becomes:

$$\begin{aligned} &\min \mathbf{c}^\top \mathbf{x} \\ &\text{s.t.} \mathbf{Ax} \geqslant \mathbf{b}, \quad \mathbf{x} \in \mathbb{R}^n \\ &\alpha^\top x \geqslant \beta, \quad x_j \in \mathbb{Z}, \forall j \in I \end{aligned} \tag{3}$$

As shown by the new formulation, cuts reduce the LP solution space, which could lead to a smaller tree in the branch-and-cut algorithm so that the number of nodes to be searched is significantly reduced. As mentioned before, the cutting plane can be combined with the branch and bound algorithm, which constitutes the branch-and-cut framework, one of the most commonly used algorithms in modern solvers. The reason is that by applying the cutting plane purely to MILP, adding cuts recursively, the resulting LP may become very large and slow to converge. Many families of valid inequalities developed, such as Chvatal-Gomory cuts (IPs) and Gomory's fractional cuts.

2.1.2. Decomposition

The branch-and-bound and cutting plane methods described above are widely used methods for solving MIPs. But in the face of large-scale MIPs, these two methods become intractable. At this time, it is necessary to decompose the large-scale problem and

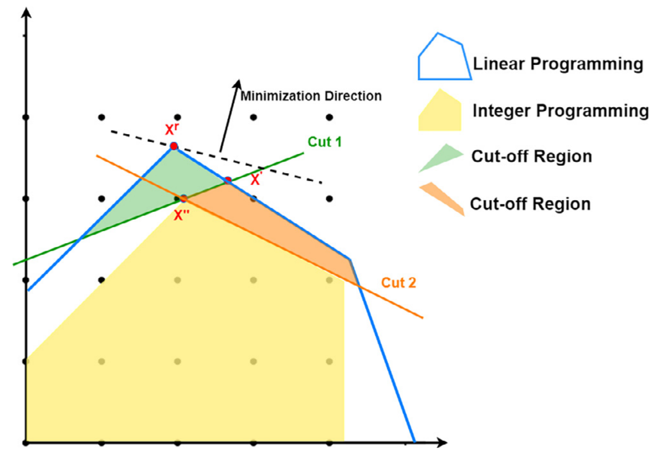


Fig. 3. cutting plane for MIP. After cutting Cut1 (green line) and cutting Cut2 (orange line), the optimal solution x'' is found.

then use the branch-and-bound and branch-and-cut methods to solve it.

The main decomposition methods are as follows: lagrangian relaxation [21], Benders decomposition [22] and Dantzig-Wolfe decomposition [23]. Lagrangian relaxation [21] is a relaxation method, which may provide better bounds than LP relaxation and can accelerate the search for the optimal solution. And lagrangian decomposition [24] is a specific form of Lagrangian relaxation. Benders decomposition [22], and Dantzig-Wolfe decomposition [22] are exact algorithms that decompose the original problem into small problems that are easier to solve and converge to the optimal solution of the original problem through iterative solutions between the small problems.

2.2. Heuristic algorithms

Heuristics play a critical role in solving MIPs. Different from exact algorithms that obtain optimal solutions at a later stage of computation, heuristics can get feasible solutions at an early stage of computation and iteratively improve existing solutions. Heuristics include two classes: construction heuristics and improvement heuristics. The construction heuristic continuously loops the following steps: fix a set of variables, strengthen the variable bound, and solve LP relaxation. The improvement heuristic chooses an initial solution and tries to improve it. In this section, we introduce four major MIP heuristics: rounding (construction), diving (construction), large neighborhood search (improvement), and feasibility pump (improvement). The idea of these primal heuristics often inspires the design of machine learning algorithms.

Rounding [25] is often a basic component of heuristic algorithms. When solving the LP relaxation of MILP and obtaining at least one linear feasible point, which is probably a fractional solution, round it to an integer solution, for example, simple rounding always keeps LP-feasibility and rounds LP-feasible solution to an integer solution while staying LP-feasible.

Diving [25] simulates depth-first search (DFS) in branch-and-bound trees with special branching rules. In the branch-and-bound process, the integer variables are fixed step by step by continuously branching and solving the relaxed LP problem until all constraints are satisfied, or the problem is inoperable. The target of diving heuristics is "quickly go down" the branch-and-bound trees, which are different from the branch and bound for the global search. If the branch of a variable is infeasible, only one level of backtracking is allowed. Various diving strategies differ in selecting the variable that should be bounded. In fractional diving, a variable closest to the integer is bounded to the nearest integer.

Large neighborhood search (LNS) [26] is a local improvement heuristic. For a problem with a large number of variables, we define a relatively large neighborhood of some incumbent solution according to some neighborhood definitions like fixing variables and adding constraints; and the neighborhood is a sub-problem of the original MIP. Then, we can apply some methods to do some searches in the sub-problem. LNS includes relaxation enforced neighborhood search (RENS) [27], relaxation induced neighborhood search (RINS) [28], local branching [29], Dins [30].

Feasibility pump (FP) [31] is a well-known primal heuristic algorithm that runs the following steps: 1) Find the rounded optimal continuous relaxation solution of the MIP (degenerating to LP); 2) Search for the nearest solution in the relaxed feasible region; 3) Perturb and round the new solution found at each step until the solution is feasible. If the limit of the maximum number of steps is reached, the algorithm will halt and return the current feasible solution as the output. The basic steps of the FP algorithm are shown in Algorithm 4.

Algorithm 4: Feasibility Pump (credit to Algorithm 1 of [32])

```
   Input: $x^0 \leftarrow \arg\min\ c^\top x$; $\bar{x}^0 \leftarrow round(x^0)$; $i = 0$
1  while $\bar{x}^i$ is not feasible do
2      $x^{i+1} \leftarrow \arg\min\ \|x - \bar{x}^i\|$
3      $\bar{x}^{i+1} \leftarrow round(x^{i+1})$
4      if $\bar{x}^{i+1} == \bar{x}^i$ then
5          random perturbation of $\bar{x}^i_j, \forall j \in I$
6      end
7      else
8          $k \leftarrow k + 1$
9      end
10 end
   Output: $\bar{x}^i$
```

There are two stages in FP: rounding and projection. The rounding stage is from LP-feasible (probably fractional points) to integer points, and the projection stage is from integer points back to LP-feasible points.

Feasibility pump includes a variety of heuristics, such as basic feasibility pump [31], objective feasibility pump [33], nonlinear feasibility pump [34].

In general, the above traditional methods provide strong priors or framework for developing learning-based techniques.

3. ML algorithms for exact solving

3.1. Branch-and-cut based

The challenge to formalize B&B resides in its inherent exponential nature: millions of Branching Variable Selection (BVS) decisions could be needed to solve a MILP, and a single bad one could result in a doubled tree size and no improvement in the search. Such a complex and data-rich setting, paired with often a lack of proper understanding of the particular problem structure, makes B&B an appealing ground for machine learning techniques, which have lately been thriving in discrete optimization. Employing a clever branching rule is critical for MIP solvers.

The majority of works focus on the decision in the B&B algorithm [58,38]. The two most crucial ones are branching variable selection and node selection. Recent works explore "learning to branch", including learning for branching variable selection [38,59,60,35,37,40,44,45,47–49,46], node selection [14,61,59,50], and cutting plane [52,51,53,54]. In particular, branching variable selection is the mainstream of the presented articles since selecting the following variable is significant in B&B and matters to the total cost of B&B.

3.1.1. Branching variable selection based

Here we introduce BVS-based machine learning methods, including supervised learning and reinforcement learning.

Naturally, the idea of adopting an imitation learning [62] strategy to learn a fast approximation of branching rules came into being. Imitation learning aims to use the expert experience to learn neural networks, which can acquire the neural networks that can reach the same or even better performance than the initial expert. [43,63,38,35,36,47] learn branching policies by imitating the strong branching rule. This approach adopts a high-quality but expensive heuristic scheme, the earliest and most widely studied method for machine learning. Table 1.

Specifically, [43] is the first to apply machine learning in the context of branching in B&B and use a classic supervised learning model to approximate full strong branching decisions. To overcome the vast computational overhead of full strong branching, they use a neural network to learn an approximated function of the variable score through handcrafted features. They can further adapt it to select the suitable variable in B&B. [38] extends [43]'s work by designing a novel pipeline that can solve the MIP on the fly. In the first 500 branches, they use the traditional SB to make decisions while recording the problem features to train the neural networks, and the neural networks take control of the decision-maker from the 501st branch. Both imitate SB successfully, while the spending time is too high due to the calculation of a large number of features on each node.

To overcome complex feature calculation, the work [35] models a MIP problem as a bipartite graph and then uses a graph-convolutional network (GCN) [64]to extract the features efficiently by various message passing approaches. The learned model is integrated into the existing solver, replacing the branch module in MIP solvers. The authors formulate MIP as a natural bipartite graph representing variable-constraint relationships, illustrated in Fig. 4. Each MIP has two nodes, one representing variables and another representing constraints. An edge between a variable *i* and a constraint *j* means variable *i* appears in constraint *j*. Much subsequent work has referenced the bipartite graph model proposed by [35].

The work [41] follows the bipartite graph encoding and utilization of the GCN model and further explores the use of machine learning to improve primal heuristic and branching strategies. This paper applies to learning the two critical sub-tasks of a MIP solver, generating an initial feasible solution and branching, and constructs two corresponding neural network-based components: neural diving and neural branching. Neural diving has the same function as the diving heuristic in the primal heuristic but has a different implementation. A conditional generative model directly generates a set of partial assignments for integer variables only at the root node of the B&B tree, thereby reducing the size of the problem and then solving the sub-problems through a solver (such as SCIP). For neural branching, it imitates an ADMM-based policy for branching. On large-scale real-world application datasets and MIPLIB, the trained neural solver achieves considerable improvement in the primal–dual gap compared with SCIP. [48] suggests a

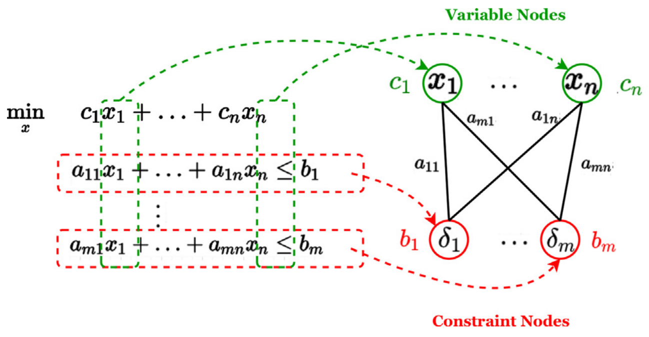


Fig. 4. Transform an MIP instance to a bipartite graph. The variables (in green) correspond to one side and the constraints (in red) refers to the other. (Redrawn while original credit to Fig.3 of [41]).

post hoc method for Neural Diving from [41] that assigns variables by confidence threshold technique, yielding better primal solutions. [36] is an improvement work on [41]. This paper proposes a hybrid architecture that uses a graph neural networks (GNN) [65] model only at the root node of the B&B tree and a weak but fast predictor at the remaining nodes, such as a simple Multi-Layer Perception (MLP). Because modeling each node as a bipartite graph in the GNN model is computationally expensive while modeling each node as a variable feature vector is computationally cheap. The model combines the superior representation framework of [35] with the computationally cheaper framework of [38] to realize a time-accuracy trade-off in branching. Even without GPU acceleration, the model still maintains good performance in terms of time cost. [47] also tries to decrease the size of the B&B tree time cost of branching GNN. The authors observe a look-back phenomenon in strong branching: the best choice for a child node is usually the second-best choice for a parent node. To better mimic strong branching, they incorporate lookback into GNN.

Unlike the bipartite graph representing, [42] generates a tripartite graph from its MIP formulation, as illustrated in Fig. 5. Based on the tripartite graph embedding, the GCN model can extract the correlation between variables, constraints, and objective functions and predicts variable solutions on the collected features. The limitation of the framework is that it only predicts values of binary variables due to the difficulties in predicting general integer vari-

Table 1
Summary of methods that combine machine learning with B&B. "Selection" denotes which part involves learning.

Method	Selection	Learning	Network	Representation	Remark
[35]	Variable	Reinforcement	GCN	Bipartite graph	Imitate strong branching
[36]	Variable	Supervised	GCN	Bipartite graph	Accelerate via dynamic embedding
[37]	Variable	Reinforcement	PD policy	Subproblem set	Evolution strategy for training
[38]	Variable	Supervised	SVM	Handcraft	Learning to rank
[39]	Variable	Supervised	GCN	Bipartite graph	Combined with DFS
[40]	Variable	Reinforcement	MLP	Handcraft	Imitate strong branching
[41]	Variable	Reinforcement	GCN	Bipartite graph	Imitate strong branching
[42]	Variable	Supervised	GCN	Tripartite graph	Extract connection information
[43]	Variable	Supervised	ExtraTrees	Handcraft	Imitate strong branching
[44]	Variable	Reinforcement	DQN	Dueling network	Learn from scratch
[45]	Variable	Reinforcement	DQN	Bipartite graph	Tree Markov Decision Processes
[46]	Variable	Reinforcement	GCN	Bipartite graph	Mix demonstration and self-generated data
[47]	Variable	Reinforcement	GCN	Bipartite graph	Imitate strong branching
[48]	Variable	Supervised	GCN	Bipartite graph	Fix variables by confidence threshold
[49]	Variable	Reinforcement	Transformer	Handcraft	Imitate reliability pseudocost branching
[14]	Node	Reinforcement	Standalone	Standalone	Imitate optimal oracle
[50]	Node	Supervised	MLP	Handcraft	Imitated the SCIP baseline
[51]	Cutting	Reinforcement	Attention & LSTM	Handcraft	Evolution strategy for training
[52]	Cutting	Supervised	MLP	Handcraft	Applicable to large-scale problems
[53]	Cutting	Supervised	GCN	Tripartite graph	Imitate look-ahead rule
[54]	Cutting	Supervised	Random forest	Handcraft	Whether to use local cuts or not
[55]	Decomposition	Supervised	Binary classifier	Handcraft	Decomposition ranking
[56]	Decomposition	Supervised	SVM & SVR	Handcraft	Detect potential decompositions
[57]	Decomposition	Unsupervised	GNN & LSTM	Bipartite graph	Learned parallel Lagrange decomposition

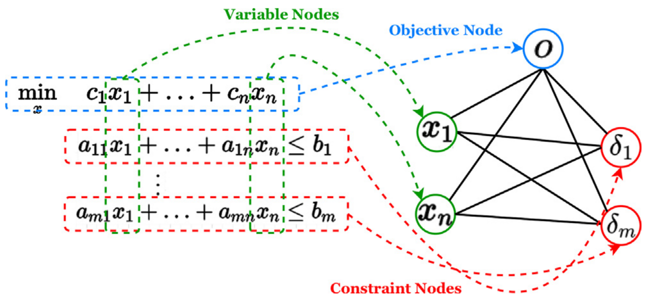

Variable Nodes
Objective Node
Constraint Nodes
$\min_x \quad c_1x_1 + \ldots + c_nx_n$
$a_{11}x_1 + \ldots + a_{1n}x_n \leq b_1$
$a_{m1}x_1 + \ldots + a_{mn}x_n \leq b_m$
O
x_1
x_n
δ_1
δ_m

Fig. 5. Converting MIP to a representation based on a tripartite graph. Three types of vertices represent objective functions (in blue), variables (in green), and constraints (in red). (Redrawn while original credit to Fig.1 of [42]).

ables. Thus, the model is more suitable for solving binary variable-intensive problems.

Furthermore, there are a few techniques [40,49] for learning to solve heterogeneous MIP's by parameterizing B&B search trees. In [40], the authors claim that information in the global branch-and-bound tree state is an essential factor in variable selection, and they design two DNN modules that extract features, including high-level branching factors and tree state information of varying sizes. Different from the previous papers that imitate strong branch, this paper imitates the relpscost branching rules in SCIP, which combines strong branching and pseudo-cost branching and is dependent on high-level branching factors. Because of this hand-crafted high-level input feature space (representation of the branch-and-bound tree rather than that of just a single node), the possibility of the model overfitting to superficial regularities is significantly decreased, achieving generalizability between instances. [49] proposes a transformer-based branching framework that evaluates mutual connections between candidate variables by the self-attention mechanism, integrating the history of branching nodes in feature extraction. Fig. 6.

Unlike learning the output of a computationally expensive heuristic used in B&B, [44] applies reinforcement learning (RL) for BVS from scratch, free from any heuristic. The core idea of RL is to learn from the interactions between the agent and the environment, and here the MIP instance is the environment, and the agent tries to solve it. Experience shows that, in some cases, the RL approach works better than the imitation learning mentioned above. In [37], the authors argue that strong branching is not a good expert to imitate because of its poor decision quality and propose to leverage RL by modeling the variable selection process as a Markov Decision Process (MDP). They further design a policy net based on primal–dual iteration over reduced LP relaxation, which utilizes the power of evolution strategy to update the neural networks. [45] revisits the work of [44] and propose tree Markov Decision Processes, which provides a more suitable framework for learning to branch. Unlike offline RL algorithms, [46] proposes a novel RL-based branching algorithm. The model trains on demonstration data at the early stage to accelerate the learning process. Then, the model updates with a mixture of demonstration and self-generated data to balance the exploration and exploitation. And a double DQN is used to improve training stability.

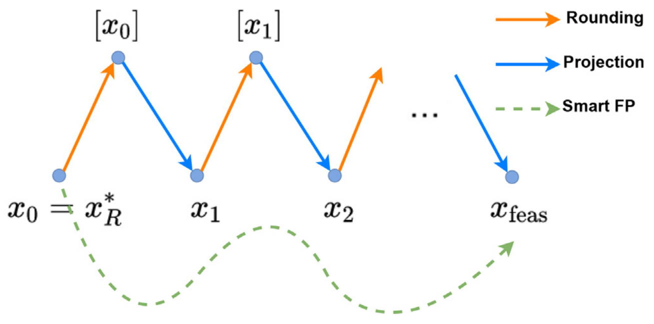


Fig. 6. The feasibility pump. (Redrawn while original credit to Fig.1 of [32]).

Based on the tree structure of the B&B algorithm, graph neural networks are widely used in problem encoding and feature extraction. Moreover, a number of BVS-based models imitate strong branching algorithms of traditional algorithms and achieve significant results. While reinforcement learning is more often used on homogenized instances. We note that the experimental dataset of [35,44,36,37,45] is randomly generated by modifying a preset backbone, thereby generating homogeneous data.

3.1.2. Node selection based

Compared with the number of BVS-based methods, there is less work in node selection of B&B. [14] uses imitation learning to train a node selection and pruning policy to speed up the tree search in the B&B process. The first is used to enforce a linear priority on the current nodes of the B&B tree, and the second is used to shrink the list of nodes by pruning further. Both adaptive strategies work together to achieve encouraging performance. Following the above work, [50] obtains a node selector by imitation learning. The difference is that [50] learns only to choose a node's children it should select. This strategy encourages finding solutions quickly instead of learning a BFS-like method (Table 2).

3.1.3. Cutting plane based

When the B&B tree is huge and with rare pruning, applying cutting plane is a useful method. cutting plane introduce tighter constraints that cut off particular relaxation solutions. Due to the importance of the cutting plane in MIP solvers, researchers try to utilize machine learning technologies to improve the traditional cutting plane algorithm. Unlike B&B, there are few well-designated traditional algorithms in the cutting plane algorithm, which denotes that imitation learning or supervised learning cannot be directly applied here. Therefore, researchers have begun to think of reinforcement learning.

[51] is the first work that focuses on learning to select cuts. The paper introduces an MDP formulation for the problem of sequentially selecting cutting plane for MIP and training a reinforcement learning (RL) agent as a subroutine in branch-and-cut. This work shows the ability of RL to improve the cutting plane algorithm and potentially opens a new research topic. The model can reduce the number of cuts required to find the optimal integer solution for small instances that can be solved using cutting plane alone. For larger-scale instances that can not be solved in less than 50 cuts, trained RL agents achieve better performance in tightening the LP relaxation. A big problem with this RL model is that it is infeasible to apply on large-scale instances. The reason is that the network inputs include all the constraints and available candidate cuts, which can incur unaffordable computational costs when applied to large-scale instances. The following work [52] proposes a cut ranking method for cut selection in the settings of multiple instance learning and solves the problem of large-scale instance application. They deploy Cut Ranking on Huawei's proprietary industrial large-scale MIP solver for production planning problems with more than 107 variables and constraints. [53] imitates a look-ahead cut selection rule and selects a single cut at a time. The look-ahead selection rule is a costly but valuable SB-like approach, where the goal is to select the cut that maximizes the improvement of the dual bounds. The model encodes the system as a tripartite graph whose nodes hold vectors (features) representing variables, constraints, and available cuts. New works are emerging recently, offering new directions. [54] solves the crucial question for applying the cutting plane whether to cut at internal nodes of the B&B tree (local cuts) or to limit all cutting activity to the root

Table 2
A brief summary of methods that combine machine learning with heuristic algorithms. The column "Strategy" denotes the combined heuristic algorithm. LNS means large neighborhood search and FP means feasibility pump.

Method	Strategy	Learning	Neural Network	Representation	Remark
[66]	LNS	Reinforcement	MLP	Handcraft	Combined with GUROBI
[67]	LNS	Reinforcement	GCN	Bipartite graph	Consider computational expense
[68]	LNS	Reinforcement	GCN	Bipartite graph	Use imitation learning
[69]	LNS	Reinforcement	GCN	Bipartite graph	Outperforms SCIP
[70]	LNS	Supervised & RL	GNN	Bipartite graph	Adaptive neighborhood size
[32]	FP	Reinforcement	MLP & CNN	Parameter matrix	Capture hidden structure of constraint matrix
[71]	P&P	Supervised	logistic regression	Handcraft	Learn to schedule heuristics
[72]	P&P	Reinforcement	Standalone	Handcraft	Pick heuristics in CPLEX
[73]	P&P	Supervised	GCN	Tripartite graph	Predict solution value for variables
[74]	P&P	Unsupervised	Clustering	Handcraft	Dynamic heuristic switching
[60]	P&P	Unsupervised	Clustering	Handcraft	Dynamic heuristic switching
[75]	P&P	Supervised	KNN	Handcraft	As a warm start
[76]	P&P	Reinforcement	Transformer	Attention layers	Early fix variables
[77]	P&P	Supervised	Random Forest	Handcraft	Optimizing solver parameters

node only. Specifically, [54] represents a MIP instance with 32 preset features and trains a regression forest to predict the speed-up (or slow-down) provided by local cuts.

The cutting plane method has been applied very successfully in practice. Even facing a large instance where the optimal solution is hard to obtain, the cutting plane can tighten the bound of the optimal value.

3.2. Decomposition based

When there are a large number of variables, cutting plane are difficult to obtain. It will be particularly useful if we can decompose a large problem into many smaller subproblems. The decomposition is highly dependent on the structure of the original model, and finding a high-quality decomposition is difficult. Therefore, decomposition-based solvers, such as GCG [78] only work for a small subset of MIPs. The core problem of a decomposition-based solver is to detect and select a high-quality decomposition.

Recently, some works consider using machine learning methods to rank decomposition. [55] trains various classifiers to decide whether to use decomposition and which decomposition to choose. Experiments show that decomposition-based solvers are promising for solving structured instances. While the detection of the potential decomposition is time-consuming. To overcome the problem of time-consuming detection, [56] investigates how detection itself can be performed and designs a data-driven framework to rank decomposition quality considering a priori features of the MIPs as predictors. In [79], the authors analyze the effectiveness of various ML methods for ranking the decomposition.

[57] considers a binary decision diagram (BDD) [80] to represent the decomposed subproblem following previous work [81–84]; and develops massively parallel Lagrangian decomposition for solving 0–1 integer linear programs, making the scheme friendly to GPU acceleration. The Lagrangean dual problem is encoded on a bipartite graph. The nodes correspond to features for primal variables and subproblems. And the edges correspond to dual variables (Lagrange multipliers). In each round of dual optimization, a GNN predicts the parameters and non-parameters, which improves the dual update of [85].

Recently, more and more work has focused on decomposition-based solvers, indicating that it is very promising, which have great promise in solving specific structural problems and parallelizing computations.

4. ML algorithms for approximate solving

Exact algorithm-based methods are concerned with bounds when solving problems and try to obtain optimal solutions, especially B&B-based methods. It is time and resource-consuming due to MIP's NP-hard nature. In many cases, especially large-scale problem instances, obtaining optimal solutions becomes intractable. Thus, heuristic-based methods are considered instead, especially in practice. Heuristic algorithms aim to solve MIP approximately by integrating a greedy approach and searching, continuously optimizing the solution. The common heuristic algorithms for MIP include: large neighborhood search (in Section 4.1) and the feasibility pump (in Section 4.2). Additionally, some works aim to better utilize existing MIP solvers or optimize decisions in existing heuristics and strategies by machine learning. We put these methods in Predict and Pick Section 4.3 and made a certain classification for some of them.

4.1. Large neighborhood search based

Large neighborhood search (LNS) [26] is a powerful heuristic for MIP. Given the problem instance and the initial feasible solution, LNS searches for better candidate solutions among pre-defined neighborhoods of the current solution in each iteration. The iteration continues until the search budget (for example, the computing time) is exhausted. This approach is easy to implement since it can leverage existing state-of-the-art MIP solvers as a black-box subroutine and does not require deep integration into solvers. It can also be said that LNS learns to identify critical substructures that are easy to solve by existing solvers.

Due to the nature of LNS, it is crucial to prevent the search from falling into a poor local optimum. In general, the size of the neighborhood grows exponentially as the size of the input problem increase. Therefore, optimizing the LNS algorithm by learning techniques is necessary to improve its efficiency.

There are two critical choices to determine the effectiveness of LNS: 1) initial solution and 2) search neighborhood at each iteration. Given an initial solution, the neighborhood selection strategy sequentially "destroys" and "repairs" the current solution iteratively. Formally, define X as the set of all possible solutions of an optimization problem $\mathscr{P}$. In the "destroy" stage, given a solution $x \in X$, a neighborhood function $N(x) \subset X$ generates a collection of candidate solutions to replace x. Then, in the "repair" stage, invoke an existing MIP solver to find the optimal solution within $N(x)$. Usually, researchers focus more on the "destroy" stage because the "repair" stage can directly invoke the existing solver.

[66] learns a neighborhood selection policy using imitation learning and reinforcement learning (RL). The model predicts a partitioning of the integer variables and decomposes the original problem into a series of sub-problems that MIP solvers can solve. Their "destroy" policy partitions the integer variables into disjoint subsets and iteratively selects one subset to optimize at a time. The decompositions for imitation learning training are generated by a random neighborhood selection policy, not depending on expert

knowledge. Following [66,67] also trains RL to learn a "destroy" policy that repeatedly unassigns one variable at a time. This setting results in the model that can only be applied to relatively small-sized instances. Following the neural diving idea proposed by [41] mentioned in Section 3.1.1, [68] adapts neural diving to obtain the initial solution and selects the search neighborhood at each LNS step. Some researchers [70] conduct analysis of LNS, and it turns out the size of neighbors is essential, and the most suitable neighbor size varies over iterations. Therefore, they propose using machine learning to find the suitable neighbor size automatically. [69] combines the ideas of the GCN feature extractor from [35] and neighborhood selection policy from [66], where the action of RL is to select the variable to be replaced. Compared with other LNS-based models, [69] can better extract a set of input features of variables and constraints.

LNS aims to continuously improve a solution, a common idea in solving CO problems. Therefore, we will discuss some approaches in the CO field that shares similar ideas with LNS, which we hope could inspire the adaptation of these methods to solve MIP. [86] proposes a framework called local rewrite, which tries to improve a given solution by selecting a part of the solution and modifying it. In their paper, the local rewrite framework is proved to be a powerful method for the vehicle routing problem and computing resource allocation. The ECO-DQN framework [87] re-designs the action space of the reinforcement learning agent, which allows revoking the previous action. In other words, the agent's action is revocable in the ECO-DQN framework. Many other works [88,89] follow the idea of local rewrite and ECO-DQN in combinatorial optimization.

4.2. Feasibility pump based

Feasibility Pump is a popular heuristic for finding a feasible solution for MIP problems and is also efficient when dealing with tricky instances. The success of the feasibility pump verified the effectiveness of decomposing the continuous relaxation and integrality.

Though FP is a relatively powerful heuristic, its efficiency and cost-effectiveness are unsatisfactory. Therefore, recent works [32] try to utilize RL to improve FP, finding feasible solutions for MIPs more efficiently. The entry point of RL is to find the next non-integer solution in step 2) in Algorithm 4. [32] presents the formulation of the FP method as an RL problem. The smart feasibility pump-CNN (SFP-CNN) works without the projection to feasible regions of the current solution and relieves the burden of calculating the projection of the current solution as the input at each time step. Instead of choosing the nearest solution, the authors in [32] let the RL agent choose the following solution as its action. Besides, two methods of MLP and CNN are designed on the representation of the state space, where the CNN is to treat the parameters $([\mathbf{A},\mathbf{b}],\mathbf{A},\mathbf{b}$is from constraint matrix$\mathbf{Ax} \geqslant \mathbf{b})$ in MIP as an image and process it. Fig. shows the FP algorithm and the smart FP proposed by [32].

4.3. Predict and pick

Instead of solving the MIPs directly, some works aim to predict and pick from the existing MIP solvers, heuristics. It is reasonable to solve MIPs based on existing works since existing MIP solvers have been improved for decades. Some researchers believe that better utilization of existing MIP methods is a valuable research topic.

Some works focus on optimizing the use of heuristics since the performance of heuristics is problem-dependent, and the selection and scheduling of heuristics are worth studying. [71] collects data on the heuristic's success across different instances and trains a binary classifier that predicts whether a heuristic will succeed at a given node. The model is an online model, meaning that during the prediction process, the model only knows the information of the current node and the branch tree without any knowledge of the remainder of the tree. By adopting reinforcement learning, [72] proposes to select the predefined heuristics in CPLEX by the features of the given instance, which is an online learning framework. [73] predicts a solution value for a subset of its binary variables based on historical data and decides whether to use heuristic algorithms or the exact branching approach to solve it. The authors try to make the exact branching approach focus on the hard case while leaving the easy case to the heuristic algorithm. Following the idea of dynamically selecting branching heuristics based on certain features of the current subproblem, [74] boosts the CPLEX MIP solver when solving set partitioning problems. A traditional model-based machine learning approach is proposed to switch between several branching heuristics. Specifically, they propose a method that first clusters the training instances based on features and finds an exemplary assignment of heuristics to clusters. Then, whenever the solver arrives at a new search node, the nearest cluster and the corresponding heuristic strategy is computed based on the current node's features. [60] uses a similar approach to [74], expanding the research from the set partitioning problem with problem-dependent heuristics to the much more general problem of MIP.

Some works combine prediction in MIPs and MIP solvers. [75] focuses on solving the MIP instances in a data-driven manner. Combined with existing MIP solvers, they train a KNN to predict redundant constraints, good initial feasible solutions, and affine subspaces where the optimal solution is likely to lie, which can lead to a significant reduction in the problem size of MIP. They conduct experiments on the electric grid unit commitment problem, which is an application of MIP. Besides, the early stopping technique is widely adopted in hyperparameter optimization [90], which uses machine learning to predict when to stop the searching process without losing quality too much. It dramatically improves efficiency and saves computing resources. The early stopping technique might be integrated into the MIP solvers. Specifically, one can use machine learning techniques to predict when to stop the B&B process early; meanwhile, the incumbent primal solution is still acceptable. [91] uses machine learning techniques to predict algorithm runtimes for problem-specific instance features and solver parameters. [76] is the first to propose the framework to accelerate approximate methods by early fixing some variables, without significantly harming the solution quality. The authors formulate the process as a Markov decision process, making variable fixing decisions sequentially. Due to the attention mechanism applied in the model, the model can only be trained on small instances and generalized to large instances.

Some works are devoted to learning to tune the parameters of MIP solvers. [77] proposes Sequential Model-based Algorithm Configuration (SMAC), a Bayesian optimization using random forests as surrogate models. Random forest is good at dealing with discrete-valued input, so SMAC solves the situation that the parameter type cannot be discrete in the traditional Bayesian optimization based on the Gaussian process. The experiments show that SMAC obtains better results than the CPLEX tuning tool [92].

Moreover, [93] manages to integrate integer programming solvers into neural network architectures as a differentiable layer capable of learning both the cost terms and the constraints. The resulting end-to-end trainable architectures can extract features from raw data and simultaneously learn a suitable set of constraints that specify any combinatorial problem. This architecture can learn to fit the right NP-hard problem needed to solve the task, demonstrating the importance of MIP in combinatorial optimization.

The above methods provide us with more ideas on how to use machine learning to solve MIPs. We can see a variety of combinations of machine learning with traditional heuristics and modern solvers.

5. Popular solvers and benchmarks

5.1. MIP solvers

Most of the modern MIP solvers are based on the branch-and-bound method, and the cutting plane technique is often integrated into the branch-and-bound method to increase the efficiency of the branch-and-bound algorithm. These two techniques constitute the branch-and-cut framework. The common MIP solvers are SCIP [94], CPLEX [95] and GUROBI [96]. Due to space limitations, this paper will not cover classical methods used in commercial solvers for solving MIP. There are emerging open-source libraries to facilitate the research at the intersection of machine learning and mixed-integer programming. MIPLearn[2] offers a customized library for learning-based solver configuration, which supports GUROBI and CPLEX. It can be framed as a borderline case of an MDP framework which is the focus of another emerging library Ecole [97]. It is designed to cooperate with the state-of-the-art open-source MIP solver SCIP where Ecole acts as an improvement to existing solvers. Besides, ORGym [98] and OpenGraphGym [99] are Gym-like libraries for learning heuristics for a collection of combinatorial optimization problems, including MIP.

5.2. Benchmarks

There is already a set of MIP benchmarks, including MIPLIB 2017 [100], MIPLIB 2010 [101], and CORLAT [102]. MIPLIB is a collection of real-world MIP instances from various academic and industrial applications, suitable for benchmarking and testing MIP solution algorithms. The MIPLIB 2017 [100] collection contains 1065 instances. A subset of 240 instances was explicitly selected for benchmark solver performance, and the selection of these instances was subject to various constraints regarding solvability and numerical stability. MIPLIB 2010 [101] is a predecessor of MIPLIB 2017, which comprises 361 instances sorted into several groups. MIPLIB 2010 includes the main benchmark test set of 87 instances, all solvable by today's codes, and the challenge test set with 164 instances, many of which are currently unsolved. CORLAT [102] is a small-scale dataset of 2000 MILP instances related to wildlife management.

6. Further discussion and outlook

Since MIP is an NP-hard problem, it is difficult to obtain an exact solution. Leveraging machine learning techniques to obtain an acceptable solution within limited computing resources is welcomed and reasonable in practical applications.

Promising future directions for research:

- Large-scale problem. There are various large-scale MIP problems in industrial scenarios, and solving them is challenging. We can deal with large-scale problems by controlling the computational overhead and reducing the problem space. More specifically, the extracted features should be independent of the size of the problem. We can consider some high-level structural features, feature aggregation, which takes various local features from instances and creates global feature vectors. In problems with many binary variables, we can reduce the solution space by fixing and unfixing these binary variables. With the help of the run-time prediction of instances, we can choose the best branching strategy or an appropriate branching variable to simplify the problem further.
- Warm start. For MIP solvers, the warm start input is usually the origin solution (possibly a partial solution), preferably already feasible. A warm start can help reduce the time to get the optimal solution. Presolve can be used to shrink the problem. Moreover, the value of a feasible solution will give the upper bound (in the case of minimization), and the upper bound can be used to prune parts of the search tree. When we have historical solutions for a specific problem, usually for new instances, starting from historical solutions can significantly improve computational efficiency. For strategies that rely on historical data to overcome cold start, we can first consider strategies that do not rely on historical data to make decisions and then switch after obtaining sufficient data. There are various ways of combining existing solvers and models, and there is still much room for exploration.
- Combine with MIP solvers. Combining with existing MIP solvers enables more flexible implementation of the model, improving the industrial usability of the model. Straightforwardly, the large-scale neighborhood search can be directly paired with existing MIP solvers to solve downstream tasks, making LNS unappealing from a deployment perspective. In addition, there are more in-depth integration methods, including parameterizing MIP solvers, making predictions, and picking from the existing MIP solvers.

There are still remaining challenges to solve:

- Generalizability. We observed the experimental results of each model and found that machine learning methods can achieve significant improvements for those datasets generated by specific scenarios or according to a particular distribution. However, for the MIPLIB dataset with large internal variance, the improvement achieved by the model is not as apparent as in the dataset mentioned above. The reason is that when training the model, the model may indiscriminately learn the signals of the data itself and those high-frequency structural signals. When the training set and the testing set are the same distribution or the same type of problem, machine learning can capture the feature structure of the same problem. Otherwise not. Industrial application scenarios often face some unexpected situations, and the robustness of the model is relatively high. How to improve the generalization ability of the model requires more thinking.
- Unified benchmark. When some work experiments compare model performance, the test set is a selected subset of the entire benchmark. Experimenting on the selected subsets will lead to biased results.

7. Conclusion

This survey presents a study of the state-of-the-art machine learning approach for solving MIP to summarize existing work and serve as a starting point for future research in these areas. We find that integrating machine learning techniques and traditional operational research algorithms is a rising topic in the research field, including a combination with the exact B&B algorithms and heuristic algorithms.

[2] https://anl-ceeesa.github.io/MIPLearn.

CRediT authorship contribution statement

Jiayi Zhang: Investigation, Writing – Original draft preparation. **Chang Liu:** Investigation, Writing – Original draft preparation. **Xijun Li:** Writing – Reviewing and Editing. **Hui-Ling Zhen:** Writing – Reviewing and Editing. **Mingxuan Yuan:** Writing – Reviewing and Editing. **Yawen Li:** Writing – Reviewing and Editing. **Junchi Yan:** Writing – Reviewing and Editing.

Declaration of Competing Interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Junchi Yan reports financial support was provided by Ministry of Science and Technology of the People's Republic of China. Junchi Yan reports financial support was provided by Huawei Technologies Co Ltd.

Acknowledgements

This work was supported in part by National Key Research and Development Program of China (2020AAA0107600), NSFC (62172056, 62222607), Shanghai Municipal Science and Technology (Major) Project (2021SHZDZX0102, 22511105100), and also sponsored by Huawei Ltd.

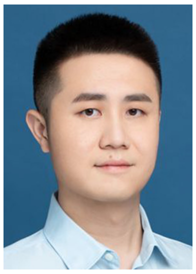
References

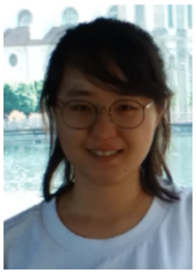
[1] Y. Pochet, L.A. Wolsey, Production Planning by Mixed Integer Programming, Springer, 2010.
[2] T. Wu, K. Akartunali, J. Song, L. Shi, Mixed integer programming in production planning with backlogging and setup carryover: Modeling and algorithms, DEDS.
[3] T. Sawik, Scheduling in supply chains using mixed integer programming, John Wiley & Sons, 2011.
[4] A.B. Keha, K. Khowala, J.W. Fowler, Mixed integer programming formulations for single machine scheduling problems, Comput. Ind. Eng.
[5] C. Malandraki, M. Daskin, Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms, Transp. Sci.
[6] T. Schouwenaars, B. De Moor, E. Feron, J. How, Mixed integer programming for multi-vehicle path planning, ECC, 2001.
[7] M. Gajda, A. Trivella, R. Mansini, D. Pisinger, An optimization approach for a complex real-life container loading problem, Omega.
[8] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, EJOR.
[9] J. Yan, S. Yang, E. Hancock, Learning graph matching and related combinatorial optimization problems, IJCAI, 2020.
[10] A. Lodi, G. Zarpellon, On learning and branching: a survey, Top.
[11] L. Huang, X. Chen, W. Huo, J. Wang, F. Zhang, B. Bai, L. Shi, Branch and bound in mixed integer linear programming problems: A survey of techniques and trends, arXiv preprint arXiv:2111.06257.
[12] A.H. Land, A.G. Doig, An automatic method for solving discrete programming problems, 50 Years of Integer Programming 1958–2008: From the Early Years to the State-of-the-Art.
[13] A.H. Land, A.G. Doig, An automatic method of solving discrete programming problems, Econometrica 28 (1960) 497.
[14] H. He, H. Daume III, J.M. Eisner, Learning to search in branch and bound algorithms, NeurIPS, 2014.
[15] D.R. Morrison, S.H. Jacobson, J.J. Sauppe, E.C. Sewell, Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning, Discrete Optimization 19 (2016) 79–102.
[16] D. Applegate, R. Bixby, V. Chvatal, B. Cook, Finding cuts in the tsp, Tech. rep. (1995).
[17] M. Bénichou, J.-M. Gauthier, P. Girodet, G. Hentges, G. Ribière, O. Vincent, Experiments in mixed-integer linear programming, Math. Programm.
[18] T. Achterberg, T. Berthold, Hybrid branching, in: CPAIOR, 2009.
[19] G.B. Dantzig, Origins of the simplex method, in: A history of scientific computing, 1990, pp. 141–151.
[20] C. Roos, T. Terlaky, J.-P. Vial, Interior point methods for linear optimization.
[21] A.M. Geoffrion, Lagrangean relaxation for integer programming, in: Approaches to integer programming, Springer, 1974, pp. 82–114.
[22] J. BnnoBRs, Partitioning procedures for solving mixed-variables programming problems, Numerische mathematik 4 (1) (1962) 238–252.
[23] G.B. Dantzig, P. Wolfe, Decomposition principle for linear programs, Oper. Res. 8 (1) (1960) 101–111.
[24] M. Guignard, S. Kim, Lagrangean decomposition for integer programming: theory and applications, RAIRO-Oper. Res. 21 (4) (1987) 307–323.
[25] M. Conforti, G. Cornuejols, G. Zambelli, Integer Programming, Springer Publishing Company, Incorporated, 2014.
[26] D. Pisinger, S. Ropke, Large Neighborhood Search (2010) 399–419.
[27] T. Berthold, Rens - relaxation enforced neighborhood search, 2007.
[28] E. Danna, E. Rothberg, C.L. Pape, Exploring relaxation induced neighborhoods to improve mip solutions, Math. Program. 102 (1) (2005) 71–90.
[29] .
[30] S. Ghosh, Dins, a mip improvement heuristic, in: International Conference on Integer Programming and Combinatorial Optimization, Springer, 2007, pp. 310–323.
[31] M. Fischetti, F. Glover, A. Lodi, The feasibility pump, Math. Program. 104 (1) (2005) 91–104.
[32] M. Qi, M. Wang, Zuo-jun, M. Shen, Reinforcement learning for (mixed) integer programming: Smart feasibility pump, in: RL4RealLife Workshop of ICML, 2021.
[33] T. Achterberg, T. Berthold, Improving the feasibility pump, Discrete Optim. 4 (1) (2007) 77–86.
[34] P. Bonami, G. Cornuéjols, A. Lodi, F. Margot, A feasibility pump for mixed integer nonlinear programs, Math. Program. 119 (2) (2009) 331–352.
[35] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, A. Lodi, Exact combinatorial optimization with graph convolutional neural networks, NeurIPS (2019).
[36] P. Gupta, M. Gasse, E.B. Khalil, M.P. Kumar, A. Lodi, Y. Bengio, Hybrid models for learning to branch, Advances in neural information processing systems 33 (2020) 18087–18097.
[37] H. Sun, W. Chen, H. Li, L. Song, Improving learning to branch via reinforcement learning, LMCA.
[38] E.B. Khalil, P.L. Bodic, L. Song, G. Nemhauser, B. Dilkina, Learning to branch in mixed integer programming, AAAI (2016).
[39] Y. Shen, Y. Sun, A. Eberhard, X. Li, Learning primal heuristics for mixed integer programs, IJCNN, 2021.
[40] G. Zarpellon, J. Jo, A. Lodi, Y. Bengio, Parameterizing branch-and-bound search trees to learn branching policies, AAAI, 2021.
[41] V. Nair, S. Bartunov, F. Gimeno, I. von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, R. Addanki, T. Hapuarachchi, T. Keck, J. Keeling, P. Kohli, I. Ktena, Y. Li, O. Vinyals, Y. Zwols, Solving mixed integer programs using neural networks, arXiv preprint arXiv:2012.13349.
[42] J. Ding, C. Zhang, L. Shen, S. Li, B. Wang, Y. Xu, L. Song, Optimal solution predictions for mixed integer programs, arXiv preprint arXiv:1906.09575.
[43] A.M. Alvarez, Q. Louveaux, L. Wehenkel, A supervised machine learning approach to variable branching in branch-and-bound, ECML, 2014.
[44] M. Etheve, Z. Alès, C. Bissuel, O. Juan, S. Kedad-Sidhoum, Reinforcement learning for variable selection in a branch and bound algorithm, in: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, 2020.
[45] L. Scavuzzo, F.Y. Chen, D. Chételat, M. Gasse, A. Lodi, N. Yorke-Smith, K. Aardal, Learning to branch with tree mdps, arXiv preprint arXiv:2205.11107.
[46] Q. Qu, X. Li, Y. Zhou, J. Zeng, M. Yuan, J. Wang, J. Lv, K. Liu, K. Mao, An improved reinforcement learning algorithm for learning to branch, arXiv preprint arXiv:2201.06213.
[47] P. Gupta, E.B. Khalil, D. Chet'elat, M. Gasse, Y. Bengio, A. Lodi, M.P. Kumar, Lookback for learning to branch, arXiv preprint arXiv:2206.14987.
[48] T. Yoon, Confidence threshold neural diving, arXiv preprint arXiv:2202.07506.
[49] J. Lin, J. Zhu, H. Wang, T. Zhang, Learning to branch with tree-aware branching transformers, Knowl.-Based Syst. 252 (2022).
[50] K. Yilmaz, N. Yorke-Smith, xxx, Learning efficient search approximation in mixed integer branch and bound, arXiv preprint arXiv:2007.03948.
[51] Y. Tang, S. Agrawal, Y. Faenza, Reinforcement learning for integer programming: Learning to cut, in: ICML, 2020.
[52] Z. Huang, K. Wang, F. Liu, H. ling Zhen, W. Zhang, M. Yuan, J. Hao, Y. Yu, J. Wang, Learning to select cuts for efficient mixed-integer programming, Pattern Recognition.
[53] M. Paulus, G. Zarpellon, A. Krause, L. Charlin, C. Maddison, Learning to cut by looking ahead: Cutting plane selection via imitation learning, in: ICML, 2022.
[54] T. Berthold, M. Francobaldi, G. Hendel, Learning to use local cuts, arXiv preprint arXiv:2206.11618.
[55] M. Kruber, M.E. Lübbecke, A. Parmentier, Learning when to use a decomposition, in: International conference on AI and OR techniques in constraint programming for combinatorial optimization problems, Springer, 2017, pp. 202–210.
[56] S. Basso, A. Ceselli, A. Tettamanzi, Random sampling and machine learning to understand good decompositions, Ann. Oper. Res. 284 (2) (2020) 501–526.
[57] A. Abbas, P. Swoboda, Doge-train: Discrete optimization on gpu with end-to-end training, arXiv preprint arXiv:2205.11638.
[58] M.-F. Balcan, T. Dick, T. Sandholm, E. Vitercik, Learning to branch, in: ICML, 2018.
[59] J. Song, R. Lanka, Y. Yue, M. Ono, Co-training for policy learning, UAI, 2020.
[60] G.D. Liberto, S. Kadioglu, K. Leo, Y. Malitsky, Dash: Dynamic approach for switching heuristics, European Journal of Operation Research.
[61] J. Song, R. Lanka, A. Zhao, Y. Yue, M. Ono, Learning to search via self-imitation with application to risk-aware planning, in: NIPS Workshop, 2017.
[62] J. Ho, S. Ermon, Generative adversarial imitation learning, NeurIPS (2016).
[63] A.M. Alvarez, Q. Louveaux, L. Wehenkel, A machine learning-based approximation of strong branching, JOC.

[64] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907.
[65] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Networks 20 (1) (2008) 61–80.
[66] J. Song, r. lanka, Y. Yue, B. Dilkina, A general large neighborhood search framework for solving integer linear programs, in: NeurIPS, 2020.
[67] ravichandra addanki, V. Nair, M. Alizadeh, Neural large neighborhood search, in: Learning Meets Combinatorial Algorithms Workshop of NeurIPS, 2020.
[68] N. Sonnerat, P. Wang, I. Ktena, S. Bartunov, V. Nair, Learning a large neighborhood search algorithm for mixed integer programs, arXiv preprint arXiv:2107.10201.
[69] Y. Wu, W. Song, Z. Cao, J. Zhang, Learning large neighborhood search policy for integer programming, NeurIPS.
[70] D. Liu, M. Fischetti, A. Lodi, Learning to search in local branching, in: AAAI, Vol. 36, 2022.
[71] E.B. Khalil, B. Dilkina, G.L. Nemhauser, S. Ahmed, Y. Shao, Learning to run heuristics in tree search, IJCAI (2017).
[72] A. Grover, T. Markov, P. Attia, N. Jin, N. Perkins, B. Cheong, M. Chen, Z. Yang, S. Harris, W. Chueh, et al., Best arm identification in multi-armed bandits with delayed feedback, AISTATS (2018).
[73] J.-Y. Ding, C. Zhang, L. Shen, S. Li, B. Wang, Y. Xu, L. Song, Accelerating primal solution findings for mixed integer programs based on solution prediction, in: AAAI, 2020.
[74] S. Kadioglu, Y. Malitsky, M. Sellmann, Non-model-based search guidance for set partitioning problems, in: AAAI, Vol. 26, 2021.
[75] Á.S. Xavier, F. Qiu, S. Ahmed, Learning to solve large-scale security-constrained unit commitment problems, INFORMS J. Comput. 33 (2) (2021) 739–756.
[76] L. Li, B. Wu, Learning to accelerate approximate methods for solving integer programming via early fixing, arXiv preprint arXiv:2207.02087.
[77] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: International Conference on Learning and Intelligent Optimization, 2011.
[78] G. Gamrath, M.E. Lübbecke, Experiments with a generic dantzig-wolfe decomposition for integer programs, in: P. Festa (Ed.), Experimental Algorithms, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 239–252.
[79] J. Weiner, A.T. Ernst, X. Li, Y. Sun, Ranking constraint relaxations for mixed integer programs using a machine learning approach (2022).
[80] S.B. Akers, Binary decision diagrams, IEEE Trans. Comput. 27 (06) (1978) 509–516.
[81] D. Bergman, A.A. Cire, Decomposition based on decision diagrams, in: C.-G. Quimper (Ed.), Integration of AI and OR Techniques in Constraint Programming, Springer International Publishing, Cham, 2016, pp. 45–54.
[82] D. Bergman, A.A. Cire, Discrete nonlinear optimization by state-space decompositions, Manage. Sci. 64 (10) (2018) 4700–4720.
[83] J.-H. Lange, P. Swoboda, Efficient message passing for 0–1 ilps with binary decision diagrams, in: International Conference on Machine Learning, PMLR, 2021, pp. 6000–6010.
[84] L. Lozano, D. Bergman, J.C. Smith, On the consistent path problem, Oper. Res. 68 (6) (2020) 1913–1931.
[85] A. Abbas, P. Swoboda, Fastdog: Fast discrete optimization on gpu, in: CVPR, 2022.
[86] X. Chen, Y. Tian, Learning to perform local rewriting for combinatorial optimization, NeurIPS, 2019.
[87] T. Barrett, W. Clements, J. Foerster, A. Lvovsky, Exploratory combinatorial optimization with reinforcement learning, AAAI, 2020.
[88] H. Lu, X. Zhang, S. Yang, A learning-based iterative method for solving vehicle routing problems, ICLR, 2020.
[89] Z.-H. Fu, K.-B. Qiu, H. Zha, Generalize a small pre-trained model to arbitrarily large tsp instances, AAAI, 2021.
[90] A. Makarova, H. Shen, V. Perrone, A. Klein, J.B. Faddoul, A. Krause, M.W. Seeger, C. Archambeau, Overfitting in bayesian optimization: an empirical study and early-stopping solution, arXiv preprint arXiv:2104.08166.
[91] F. Hutter, L. Xu, H.H. Hoos, K. Leyton-Brown, Algorithm runtime prediction: Methods & evaluation, Artif. Intell. 206 (2014) 79–111.
[92] F. Hutter, H.H. Hoos, K. Leyton-Brown, Automated configuration of mixed integer programming solvers, in: International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming, Springer, 2010, pp. 186–202.
[93] A. Paulus, M. Rolínek, V. Musil, B. Amos, G. Martius, Comboptnet: Fit the right np-hard problem by learning integer programming constraints, in: ICML, 2021.
[94] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M.E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, J. Witzig, The SCIP Optimization Suite 8.0, ZIB-Report 21–41, Zuse Institute Berlin (December 2021).
[95] I.I. Cplex, V12. 1: User's manual for cplex, International Business Machines Corporation 46 (53) (2009) 157.
[96] Gurobi Optimization, Gurobi optimizer reference manual.
[97] A. Prouvost, J. Dumouchelle, L. Scavuzzo, M. Gasse, D. Chételat, A. Lodi, Ecole: A gym-like library for machine learning in combinatorial optimization solvers, arXiv preprint arXiv:2011.06069.
[98] C.D. Hubbs, H.D. Perez, O. Sarwar, N.V. Sahinidis, I.E. Grossmann, J.M. Wassick, Or-gym: A reinforcement learning library for operations research problems, arXiv preprint arXiv:2008.06319.
[99] W. Zheng, D. Wang, F. Song, Opengraphgym: A parallel reinforcement learning framework for graph optimization problems, in: International Conference on Computational Science, 2020.
[100] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. M. Christophel, K. Jarck, T. Koch, J. Linderoth, M. Lübbecke, H.D. Mittelmann, D. Ozyurt, T.K. Ralphs, D. Salvagnin, Y. Shinano, MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library, Mathematical Programming Computation.
[101] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R.E. Bixby, E. Danna, G. Gamrath, A.M. Gleixner, S. Heinz, et al., Miplib 2010, Math. Programm. Comput. 3 (2) (2011) 103–163.
[102] C.P. Gomes, W.-J. v. Hoeve, A. Sabharwal, Connections in networks: A hybrid approach, in: International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming, Springer, 2008, pp. 303–307.

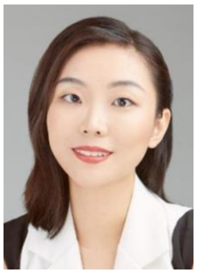
Jiayi Zhang received her B.E. degree in Computer Science and Technology from Shanghai Jiao Tong University, Shanghai, China in 2020. She is working toward the Master's degree at Department of Computer Science and Engineering, Shanghai Jiao Tong University. Her research interests include reinforcement learning and combinatorial optimization.

Chang Liu is a PhD Student with Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. He received the B.E. degree (with highest honor in SJTU-ACM Class) in Computer Science from Shanghai Jiao Tong University in 2020. His research interests include reinforcement learning and combinatorial optimization. He has published first-authored papers in ECCV 2022, ICML 2022, ICDE 2021.

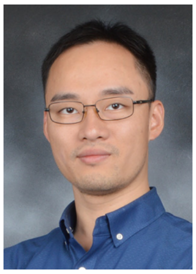

Xijun Li received the master degree from Shanghai Jiao Tong University, P.R. China, in 2018. Currently, he is a senior researcher of HUAWEI Noah's Ark Lab and working towards his Ph.D. degree in the University of Science and Technology of China (HUAWEI-USTC Joint Ph.D. Program). He has published several papers on top peer-reviewed conferences and journals (SIGMOD, KDD, ICDE, DAC, CIKM, ICDCS, TCYB, etc.). Moreover, his research interests focus on machine learning for combinatorial optimization and computer systems.


Hui-Ling Zhen is a research scientist in Noah's Ark Lab, Huawei. She was a post-doctoral research fellow in City University of Hong Kong, after she received the B.S. degree in numerical mathematics and the Ph.D. degree in applied mathematics from the Beijing University of Posts and Telecommunications, Beijing, China. Her research interests include large-scale optimization, constraint programming, and the applications in supply chain management and chip design.

Mingxuan Yuan is currently a principal researcher of Huawei Noah's Ark Lab. He received the PhD degree from the Hong Kong University of Science and Technology, and the B.E and M.S degrees both from Xi'an Jiaotong University. His research interests include data-driven optimization algorithms, data-driven SAT/MIP solving algorithms and data-driven EDA algorithm.



Yawen Li is currently an Associate Professor of School of Economics and Management, Beijing University of Posts and Telecommunications. She received her Ph.D. in Innovation, Entrepreneurship and Strategy from Tsinghua University in 2018. Her research interest focuses on artificial intelligence, collaborative innovation, the development of science parks, and scientific productivity of firms. Her research papers have been published in or accepted by journals including IEEE TKDE, Neurocomputing, Asian Journal of Technology Innovation, Journal of Leadership and Organizational Studies, etc.



Junchi Yan is currently an Associate Professor with Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. Before that, he was a Senior Research Staff Member with IBM Research where he started his career since April 2011. His current research interests include machine learning (especially for combinatorial optimization) and computer vision. He served as Area Chair for NeurIPS/ICML/CVPR/AAAI/ACM-MM and Senior PC for IJCAI/CIKM, and Associate Editor for Pattern Recognition.