2023 IEEE 20th International Conference on Mobile Ad Hoc and Smart Systems (MASS) | 979-8-3503-2433-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/MASS58611.2023.00045

Reinforcement Learning for Node Selection in Mixed Integer Programming

Sijia Zhang*†, Shaoang Li*†, Feng Wu*†, Xiang-Yang Li*
* School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
†Institute of Dataspace, Hefei, China
Email: sxzsj@mail.ustc.edu.cn, lishaoa@mail.ustc.edu.cn,wufeng02@ustc.edu.cn,xiangyangli@ustc.edu.cn

***Abstract*—Mixed Integer Linear Programming (MILP) optimization is a critical NP-hard problem applied across various sectors. Although machine-learning-based strategies have used historical data and expert strategies to effectively expedite optimal solution identification, they often neglect the importance of early search tree exploration and the cost associated with path backtracking. In this paper, we propose an innovative reinforcement learning (RL) approach for the node selection in the Branch-and-Bound (B&B) process. Our solution includes a novel reward function, an epsilon-greedy strategy for balancing exploitation and exploration, and a unique RL technique adaptable to the dynamic nature of the state and action spaces. Our empirical results show that our approach achieves an average of 10% acceleration compared to traditional SCIP and significantly outperforms other AI-based methods, underscoring the immense potential of RL in MILP optimization.**

I. Introduction

Mixed Integer Linear Programming (MILP) constitutes a pivotal optimization problem widely deployed across an array of sectors such as supply chain optimization [1], industrial process scheduling [2], facility location [3], and distributed energy systems [4]. The primary objective of MILP lies in discerning an optimal solution constrained by linear equations, wherein some or all variables must be integers. This confluence of continuous and discrete variables engenders considerable challenges, thereby classifying MILP as an NP-hard problem [5].

In addressing this inherent complexity, the Branch-and-Bound (B&B) algorithm [6] has ascended to prominence within contemporary MILP solvers [7]–[9], primarily attributed to its adaptive nature and systematic exploration capabilities. Within the B&B framework, the problem is segmented into smaller, more manageable subproblems, each represented as a node within a search tree. This bifurcation facilitates a granular approach to problem-solving, enabling the solver to concurrently and effectively manage the continuous and discrete components of the problem.

However, a salient problem within the B&B algorithm persists: the selection of the node in the search tree. A judicious node selection strategy can direct the exploration towards promising areas within the search tree, augmenting the likelihood of rapidly identifying a strong incumbent solution, which can subsequently be leveraged to dismiss other nodes [10]. In this paper, we focus on **the node selection problem**, which has a significant impact on the overall solver performance.

Modern solvers often select nodes by balancing the application of heuristic algorithms such as ESTIMATE, Depth-First Search (DFS), and Breadth-First Search (BFS) [11]. They tackle the limitations of a single heuristic by dynamically adjusting the invocation probabilities, making the most of the strengths of each heuristic while compensating for their limitations. However, this approach often heavily relies on expert-crafted strategies and requires considerable domain knowledge or extensive manual tuning.

Recent advancements in artificial intelligence (AI) have catalyzed the rise of machine-learning-based strategies in node selection. These strategies [12]–[14] primarily train models on diverse problem instances, leverage historical optimal solution paths as expert strategies, and incorporate both local and global features for expedited identification of optimal solutions within similar problem types. The ability to learn and adapt offers great promise in efficiently identifying optimal solutions while reducing the number of nodes to be explored.

However, these approaches mainly concentrate on the efficient exploitation of promising solutions, henceforth denoted as **(I1)**. Issues such as the necessity of broad tree exploration for global bound update, referred to as **(I2)**, and the cost of backtracking in the B&B tree, termed as **(I3)**, often receive less emphasis. Specifically, these methods may overlook the criticality of extensive tree exploration **(I2)** for updating global bound estimates and the cost associated with backtracking **(I3)**. This lack of attention can potentially result in the problem of local optima entrapment in the early stages of the search process, thereby affecting the overall efficiency and effectiveness of the solution procedure.

Our extensive empirical analyses underline the critical need for node selection optimization to not only consider its influence on node quantity but also its implications for the operational efficiency of other components in the solution process. For instance, **(I3)** the order of node selection can instigate a significant increase in path backtracking overhead. This overhead, an aspect often overlooked, carries substantial time costs. Specifically, the cost of path backtracking accounts

Xiang-Yang Li is the corresponding author. The research is partially supported by National Key R&D Program of China under Grant No. 2021ZD0110400 , Innovation Program for Quantum Science and Technology 2021ZD0302900, China National Natural Science Foundation with No. 62132018 , "Pioneerand""Leading Goose"R&D Program of Zhejiang", 2023C01029.

2155-6814/23/$31.00 ©2023 IEEE
DOI 10.1109/MASS58611.2023.00045

for 15.2% of the total solving time in the WPMS problem with parameter $n \in [60, 70]$. Path backtracking involves retracing the decision path from the current selection back to a common ancestor node with a previous selection. This operation could be computationally intensive, particularly in complex or large-scale problems.

In this paper, we advocate for the application of reinforcement learning (RL), an AI technique celebrated for its ability to optimize intricate decision-making processes in dynamic environments. One such environment is the evolving search tree in the B&B process, which makes RL an appealing choice. Nonetheless, the implementation of reinforcement learning in this scenario encounters several significant challenges:

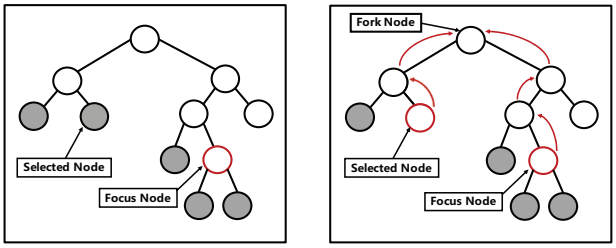
- **Reward Function Design:** One main challenge is creating a reward function that accurately reflects the short-term appropriateness of node selection and its long-term implications, especially when considering aspects like solution exploitation (I1) and path backtracking cost (I3).
- **Balancing Exploitation and Exploration:** In the B&B process, deciding when to adhere to learned policies versus exploring new nodes is a significant challenge, as this balance can affect the process's efficiency and potentially lead to local optima traps (I2).
- **Managing Dynamic Environments:** The evolving nature of the B&B process, with constantly emerging nodes, results in variable state and action spaces, challenging traditional RL models. The task is to devise an RL approach that can handle this variability and still make consistently effective decisions.

To address these challenges, we devise an innovative reward function which harmoniously marries the reward for uncovering an optimal solution with the penalty associated with path backtracking. In response to the exploitation-exploration conundrum, we adopt an epsilon-greedy strategy for node selection, thereby infusing a modicum of randomness to strike an optimal balance between reliance on learned policies and the exploration of potentially promising avenues. Finally, in cognizance of the dynamic characteristics of the B&B process, we propose a unique reinforcement learning technique. Our approach, rather than directly opting for nodes, produces a feature vector which is subsequently juxtaposed with the features of available nodes to guide decision-making.

Summary of results: We evaluate 4 AI-based node selection methods for solving MILP problems on three types of problems: FCMCNF, WPMS, and GISP. Our RL method outperforms the SCIP's ESTIMATE, DFS, and BFS methods in speed, exhibiting an average speed increase of approximately 4.29%, 2.66%, and 2.69% respectively. For FCMCNF and GISP problems, our approach provides faster solving times than all other AI-based techniques.

The salient contributions of this work are summarized as follows:

- **Highlighting the Path Backtracking Cost:** We shed light on a critical issue largely neglected in prior works - the cost implicated in path backtracking (**I3**). We emphasize that the temporal overhead associated with path backtracking is integral in the design of an effective and efficient reinforcement learning method for the B&B process.
- **Proposing a Comprehensive Solution:** We put forth a novel solution that concurrently contemplates the reward for identifying the optimal solution (**I1**), the balancing act between exploitation and exploration (**I2**), and the cost of path backtracking (**I3**). By concocting an ingenious reward function and adopting an epsilon-greedy strategy in node selection, our approach deftly navigates these factors, facilitating robust and efficient decision-making in the B&B process.
- **Demonstrating Superior Performance:** We corroborate the efficacy and superiority of our proposed method through rigorous experiments. The experimental results unequivocally attest that our approach significantly outperforms traditional methods, thereby validating the effectiveness of our proposed solution to the challenges of deploying reinforcement learning in the B&B process.



(a) (b)

Fig. 1: Path backtracking process.

In Section II, we formally introduce the B&B framework. The motivation behind our work is detailed in Section III. In Section IV, we delve into our approach, defining the node selection problem as a Markov decision process and presenting our model, state representation, and training procedures. Experimental results and their discussion are presented in Section V. Section VI contains a review of pertinent literature and a comparison with our work. Finally, Section VII encapsulates our findings and explores future research directions.

II. Preliminaries

A. Mixed Integer Linear Programming (MILP)

A Mixed Integer Linear Programming (MILP) problem is defined by a set of decision variables, where a subset or all variables are required to be integers. The objective is to minimize a linear function under a series of linear constraints, as formulated below:

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

In this formulation, $\boldsymbol{c} \in \mathbb{R}^n$ represents the objective coefficient vector, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is the constraint coefficient matrix,

$\boldsymbol{b} \in \mathbb{R}^m$ signifies the constraint right-hand-side vector, and $I \subseteq 1, \ldots, n$ denotes the indices of the integer variables. The problem's size is typically represented by the number of rows m and columns n in the constraint matrix. The MILP problem's optimum lower bound can be inferred by solving the corresponding Linear Programming (LP) relaxation of the MILP.

B. Branch-and-Bound (B&B)

The Branch-and-Bound (B&B) algorithm, a well-regarded tree search method, is commonly used to solve MILP problems. The strategy behind B&B involves a divide-and-conquer approach, breaking down the search space by branching on variable values.

To elaborate, an MILP problem, expressed as $P(X, f)$, is defined wherein $X \subseteq \mathbb{R}^n$ represents the feasible region defined by the problem constraints, and $f : X \to \mathbb{R}$ is the linear objective function. The aim is to discover the optimal $x^* \in X$ that minimizes f under the imposed constraints, with a subset or all components of x^* being integers.

The LP relaxation of the problem can be solved as illustrated below:

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

In this Equation (2), all variables are treated as continuous. Efficient algorithms such as the simplex method can be utilized to solve this equation, and the optimal solution thus obtained provides a lower bound $f(a)$ for the original MILP problem.

III. Motivation

Node selection plays a pivotal role in the B&B algorithm, with substantial ramifications on the size of the search tree and the overall efficiency of MILP solvers. An effective node selection strategy must strike an optimal balance between the following primary objectives:

- **Issue 1 (I1)**: Swift identification and exploitation of a promising solution (also known as securing a strong incumbent) to facilitate deeper exploration of the search tree. Rapid discovery of a high-quality solution permits an immediate upper bound (UB) update, aiding in the pruning of nodes possessing UBs surpassing the current optimal solution. This pruning markedly diminishes the search space, bolstering computational efficiency.
- **Issue 2 (I2)**: Broad node exploration to update bound estimates, thus fostering breadth-oriented expansion of the search tree. For minimization problems, this expansive strategy uncovers larger lower bounds (LB) through comprehensive node exploration at each level before advancing to the next. Updating the LB, which necessitates the exploration of all sibling nodes, furnishes a more accurate worst-case scenario estimate. This, in turn, aids in better assessing the quality of the current best solution, thereby enabling more informed decision-making during the search process.

The dynamic B&B tree structure, in conjunction with the ever-evolving state and action spaces in MILP problems, poses significant challenges to traditional node selection heuristics. Furthermore, contemporary solvers often resort to predetermined probabilities to trigger different heuristic search algorithms, demanding significant domain knowledge and extensive manual tuning.

A holistic node selection strategy should encapsulate not only the aforementioned factors, but also the operational efficiency of the overall solution process. This realization introduces a third issue that must be considered:

- **Issue 3 (I3)**: The order of node selection, which can precipitate a considerable escalation in path backtracking overhead. Path backtracking entails retracing the decision path from the current selection to a common ancestor node with a preceding selection, a process that could be computationally taxing, especially in complex or large-scale problems.

Our primary goal is to conceive strategies capable of dynamically adapting to search tree modifications and adeptly balancing the factors discussed above (**I1**, **I2**, and **I3**). Such strategies promise to significantly enhance the efficiency of MILP solvers, opening avenues for tackling increasingly complex and large-scale problems.

IV. Methodology

Our proposed methodology, embedded in a Reinforcement Learning (RL) framework, introduces a series of novel methods to address the challenges inherent in Mixed Integer Linear Programming (MILP) and the Branch-and-Bound (B&B) process. It primarily consists of three sections: formulating the problem as a Markov Decision Process (MDP), integrating a Reinforcement Learning (RL) strategy, and balancing exploration and exploitation using an adapted epsilon-greedy strategy. We also employ a dynamic reward function to manage the balance between swiftly identifying promising solutions and controlling the backtracking cost.

A. Markov Decision Process Formulation

The sequence of decisions made during the Branch-and-Bound (B&B) process can be conceived as a Markov Decision Process (MDP). In this perspective, we consider the solver as the environment and the node selector as the agent. The state s_t at any decision stage t encapsulates the entire B&B tree, the current best integer solution, the Linear Programming (LP) solution for each node, the node in focus, and pertinent solver statistics, such as the frequency of each primal heuristic invocation. This state representation forms the basis of our transition into a novel node feature extraction framework, which we detail in Section IV-B. Rather than following the traditional approach of pairwise node comparisons, our framework aims to achieve a generalized representation of effective nodes.

A distinct node representation function $\pi(a_t|s_t)$ is proposed, which forms a feature vector a_t by considering all open nodes in the B&B tree. This vector resides in a feature space $A \in \mathbb{R}^k$, where k is the dimension of node features. Our method then selects nodes closest to these learned features

from the candidate set. The solver subsequently expands the B&B tree, solves the LP relaxations for the selected node, invokes any internal heuristic, prunes the tree if needed, and finally bifurcates it into two child nodes. This leads us to a new state s_{t+1}, and the node selection policy is then called again to make the next decision. This process continues until the instance is solved, i.e., there are no more leaf nodes left for selecting.

$$p_\pi(\tau) = p(s_0) \prod_{t=0}^{T-1} \sum_{a \in A} \pi(a|s_t) p(s_{t+1}|s_t, a).$$

This MDP perspective differs from prior work that mainly focused on leaf node selection, similar to a depth-first search (DFS) approach where the agent chooses between only two nodes. We propose a method that selects from all open nodes, introducing a challenge: the action space is dynamic, as the number of open nodes changes throughout the solution process. Our objective to reduce path backtracking overhead necessitates the consideration of all open node features alongside the previously selected node. Consequently, our state space is also dynamic and not fixed, adding an additional layer of complexity to the learning process. We address this challenge in our node selection strategy in Section IV-C, leveraging our reinforcement learning scheme.

B. Node Feature Extraction Framework

A challenge stems from the dynamic nature of the Branch-and-Bound (B&B) search tree, which is perpetually updated, thereby producing new nodes. As a result, the size of the state and action spaces for the RL process changes variably. Under these circumstances, obtaining an action directly from the RL network may be arduous.

In our research, we make a significant departure from conventional node selection methodologies often employed in previous works, which generally concentrate solely on pairwise node comparisons. Instead, we aim to cultivate a more universal representation of successful nodes by introducing a unique node representation function. This function takes into account all open (unexplored) nodes within the Branch-and-Bound (B&B) tree, aiming to discern patterns of node features that have historically proven to be ideal for selection.

This function compiles a feature vector, capturing key attributes that are characteristic of successful nodes. Such attributes include node depth, objective value, estimated bound, and other critical properties within the search tree. These node features are derived with modifications from the work of He et al. We have removed some features that do not significantly change during the solving process.

Once these features have been identified, our method identifies the nodes in the candidate set that most closely align with these learned features for selection. Consequently, the node representation function serves as our "ideal" node representation, providing a benchmark for comparison against all open nodes.

This innovative shift in perspective from learning a *node-comp* function to learning a node representation function enables us to contemplate all open nodes simultaneously, providing a more panoramic view of the search tree. Such a strategy allows us to choose a node that aligns most with our ideal node representation, consequently leading us closer to the optimal solution.

C. Reinforcement Learning and Reward Function Design

In our research, we adopt the Deep Deterministic Policy Gradient (DDPG) [15] reinforcement learning algorithm. DDPG, an actor-critic, model-free, off-policy method, is tailored specifically to address continuous action spaces. This design is primarily motivated by the inherent characteristics of MILP problems.

During the solving process of MILP problems, the number of open nodes is dynamic. It continually changes as new feasible solutions are identified, rendering traditional reinforcement learning methods with discrete action spaces, such as Q-Learning [16] or Deep Q-Learning [17], less suitable. Moreover, the features extracted from nodes in the search tree lean more towards a continuous space, further emphasizing the appropriateness of DDPG.

Our actor and critic models both follow a standard fully-connected multi-layer perceptron structure with H hidden layers, U hidden units per layer, ReLU activation layers, and Batch Normalisation layers after each activation layer. Additionally, we implement a Dropout layer with a dropout rate of p_d.

Despite these considerations, several challenges persist. The most notable challenge pertains to the training time of the RL algorithms. Generally, the episode length is proportional to performance. As randomly initialized policies tend to perform poorly, the early stages of training with standard reinforcement learning algorithms often progress extremely slowly. This sluggish initial learning pace can extend the total training time considerably.

To address this issue, we harness instances of similar problems for pre-training. We first train the RL model on these instances, imbuing it with an offline decision-making capability. Consequently, we commence with an offline policy, learned from historical data, that presents a relatively proficient initial policy. This approach circumvents the high computational cost associated with online learning that employs inefficient initial policies.

In the context of our problem, an ideal reward function should encourage the discovery of feasible solutions and discourage unnecessary exploration that leads to backtracking and increases the computational cost. To achieve this, we design the reward function as follows: If the optimal solution is in the current node area, the reward is 1. If the optimal solution is not in the current node area, the reward is 0. Regardless of whether the position is in the area or not, we penalize backtracking by subtracting a term proportional to the number of backtracking steps from the reward.

Mathematically, the reward function R for a given node n, lower bound of the node lb_n, upper bound of the node ub_n, optimal solution x^*, backtracking steps b, and penalty factor λ is defined as:

$$R(n, x^*, b, \lambda) = \begin{cases} 1 - \lambda b & \text{if } x^* \in [lb_n, ub_n], \\ 0 - \lambda b & \text{otherwise.} \end{cases} \quad (3)$$

The penalty factor λ is a tunable hyperparameter that controls the impact of backtracking on the reward, thereby providing a flexible and tunable reward function for our problem.

D. Balancing Exploration and Exploitation

In the dynamic environment of the Branch & Bound (B&B) process for Mixed Integer Linear Programming (MILP), balancing exploration and exploitation is a complex task. Exploitation, which involves decisions based on the knowledge the RL agent has accumulated, can efficiently direct the search towards promising solutions. However, this approach risks stagnation at local optima. On the other hand, exploration introduces randomness into the decision-making process to discover new, potentially promising areas but may inadvertently overlook immediate beneficial solutions.

To strike a balance, we implemented an epsilon-greedy strategy due to its simplicity and proven efficacy, especially in an online learning context where the agent is required to learn and decide concurrently. This method enables the RL agent to frequently choose the best action, as per its existing knowledge, but occasionally (with an epsilon probability) select a random action, fostering exploration.

Our problem posed unique challenges necessitating an adaptation of the standard epsilon-greedy strategy. Firstly, every node in the B&B tree is only explored once, making redundant explorations non-existent. Second, the action space, that is, the number of open nodes (N), is dynamic and constantly changing. To cater to these specifics, we introduced a variation in our epsilon-greedy strategy: we tied the epsilon value to the size of the current action space. At the beginning of each round, the length of the open node queue, N, is determined. Subsequently, a coin is tossed with a success probability, $\epsilon_t = N^{-1/3} \cdot (\log t)^{1/3}$, where t is the current round. A success leads to exploration: a node is chosen uniformly at random. Conversely, a failure leads to exploitation - the node with the feature closest to the feature generated by the actor network is chosen.

This strategy ensures that as the action space expands, the epsilon value reduces, promoting exploitation, and vice versa. By dynamically adjusting the balance between exploration and exploitation during the B&B process, we aim to improve search tree pruning (I1) and bound estimates (I2). Thus, our RL agent becomes more efficient and effective in solving complex MILP problems.

V. Experiments

We now present experimental results on three NP-hard problems, including FCMCNF, WPMS and GISP. We evaluate each of four machine learning methods by running it in SCIP with a simple highest-priority rule. We also evaluate three default node selection rules in SCIP.

A. Benchmarks

In our evaluation process, we similarly employ three NP-hard instance families, just like Labassi et al. in the latest node selection work [14]. These families are particularly primal-difficult, that is, finding feasible solutions for them poses the main challenge. The first benchmark is composed of Fixed Charge Multicommodity Network Flow (FCMCNF) instances, generated from the code of Chmiela et al. [18]. We train and test on instances with $m = 1.5 \times n$ commodities. The second benchmark is composed of Weighted Partial MaxSAT (WPMS) [19] instances, generated following the scheme of Béjar et al. [20]. Our third benchmark is composed of Generalized Independent Set (GISP) instances [21], generated from the code of Chmiela et al. [18]. All these families require an underlying graph: we use in each case Erdős–Rényi random graphs with the prescribed number of nodes, with edge probability $p = 0.3$ for FCMCNF and $p = 0.6$ for WPMS and GISP.

B. Baselines

In our study, we compare our method against three optional node selection strategies in SCIP: ESTIMATE, DFS (Depth-First Search), and BFS (Breadth-First Search). The ESTIMATE strategy is the state-of-the-art choice typically employed. The DFS strategy, in contrast, prioritizes deeper exploration in the search tree, while the BFS strategy emphasizes breadth, exploring the width of the search tree first.

In addition, we compare against three machine learning approaches: the Support Vector Machine (SVM) approach [12], the RankNet feedforward neural network approach [22], and a state-of-the-art approach based on Graph Neural Networks (GNN) [14]. The SVM and RankNet methods utilize a multilayer perceptron; the latter varies for one benchmark where they use three hidden layers, while for simplicity, we use a multilayer perceptron with a hidden layer of 32 neurons for all benchmarks (MLP). The GNN method uniquely leverages the structure of the graph to guide node selection. The features used in these papers are roughly similar; again, for simplicity, we adopt the fixed-dimensional features of He et al. [12] for both the SVM and RankNet method.

C. Training and Evaluation

In this work, the training procedure we adopt for all machine learning models aligns closely with the methodology introduced by He et al. [12]. Our model implementations are built using the robust PyTorch framework, and we optimize them through the application of the Adam optimization algorithm.

For the FCMCNF, WPMS, and GISP problem domains, we generate extensive datasets, consisting of 20,000 training samples and 2,000 test samples. From these, we randomly select 2,000 samples for training and 500 samples for testing within each problem, providing a diverse set of scenarios for

TABLE I: Comparison of Average Solving Time and B&B Tree Size (Test).
FCMCNF: $n = 15$ nodes. WPMS and GISP: number of nodes $n \in [60, 70]$.

Methods	FCMCNF		WPMS		GISP	
	Time(s)	Nodes	Time(s)	Nodes	Time(s)	Nodes
ESTIMATE(SCIP)	2.80 ± 1.40	37.36 ± 4.57	4.92 ± 1.62	170.29 ± 2.13	2.60 ± 1.25	107.98 ± 3.16
DFS(SCIP)	2.88 ± 1.44	51.09 ± 5.91	5.25 ± 1.63	224.61 ± 2.14	2.54 ± 1.25	**97.56 ± 3.06**
BFS(SCIP)	2.70 ± 1.39	36.16 ± 4.46	4.79 ± 1.59	157.72 ± 2.02	2.65 ± 1.27	107.76 ± 3.26
SVM [12]	3.04 ± 1.45	44.31 ± 5.98	5.65 ± 1.76	198.34 ± 2.39	2.68 ± 1.30	111.32 ± 2.85
RankNet [13]	3.45 ± 1.59	45.62 ± 4.86	5.05 ± 1.60	161.79 ± 2.47	3.10 ± 1.37	108.07 ± 3.21
GNN [14]	2.77 ± 1.50	**30.69 ± 4.73**	**4.66 ± 1.89**	**157.29 ± 2.89**	2.74 ± 1.28	143.71 ± 2.79
RL(Our method)	**2.68 ± 1.47**	30.70 ± 5.31	4.79 ± 1.80	186.15 ± 2.81	**2.53 ± 1.30**	106.85 ± 3.31

TABLE II: Comparison of Average Solving Time and B&B Tree Size (Transfer).
FCMCNF: $n = 20$ nodes. WPMS and GISP: number of nodes $n \in [80, 100]$.

Methods	FCMCNF		WPMS		GISP	
	Time(s)	Nodes	Time(s)	Nodes	Time(s)	Nodes
ESTIMATE(SCIP)	13.54 ± 2.01	87.13 ± 5.99	**33.13 ± 1.79**	491.48 ± 1.69	6.60 ± 1.36	348.80 ± 2.18
DFS(SCIP)	16.29 ± 2.30	151.41 ± 8.67	40.04 ± 2.26	701.78 ± 2.21	**6.58 ± 1.30**	317.16 ± 2.23
BFS(SCIP)	13.12 ± 2.03	81.85 ± 5.78	34.57 ± 1.82	**428.20 ± 1.64**	7.64 ± 1.35	386.93 ± 1.95
SVM [12]	10.84 ± 2.06	107.09 ± 7.66	41.75 ± 1.96	576.24 ± 1.76	8.58 ± 1.41	331.17 ± 2.49
RankNet [13]	10.49 ± 2.12	72.76 ± 6.32	54.43 ± 2.38	871.27 ± 2.48	8.08 ± 1.44	362.75 ± 2.34
GNN [14]	10.24 ± 2.33	72.20 ± 7.04	63.10 ± 2.78	799.94 ± 2.77	11.76 ± 1.49	**285.42 ± 2.51**
RL(Our method)	**9.44 ± 2.02**	**60.14 ± 5.50**	45.13 ± 2.20	587.31 ± 1.81	8.80 ± 1.93	349.01 ± 2.46

our models to train and test effectively. This variety promotes the adaptability and generalization capabilities of our models. The training process consistently applies a batch size of 16 and is conducted on a MacBook Pro equipped with the Apple M1 chip, featuring an 8-core CPU and 16GB of unified memory.

The evaluation procedure used by us is common within the mixed-integer programming community: the 1-shifted geometric mean and geometric standard deviation are calculated to estimate the average and dispersion of the Branch-and-Bound (B&B) tree size and solving time across our benchmarks. This approach effectively accounts for outlier effects from overly simple or exceedingly challenging instances. Our evaluations are conducted on a set of 100 test and 100 transfer instances, as detailed in Section V-A.

D. Results and Analysis

Tables I and II illustrate the results of various methods solving MILP problems. These methods span both traditional and AI-based approaches and are tested on three types of problems: FCMCNF, WPMS, and GISP.

The results from Table I underscore that our RL method on average exhibits a speed increase of approximately 4.29%, 2.66%, and 2.69% over the SCIP's ESTIMATE, DFS, and BFS methods respectively across all three problem types. For FCMCNF and GISP problems, our method provides faster solving times than all other AI-based techniques. In the WPMS problem, although the RL method presents a larger B&B tree size, it maintains competitive solving times, affirming its efficiency.

Interestingly, our experiment also uncovers our observation: **a smaller solving time doesn't necessarily correspond to fewer nodes in the search tree**. For instance, in the WPMS problem, despite the RL method exhibiting a larger B&B tree size than GNN, it achieves similar solving times. This underlines our assertion that the impact of node selection on solving efficiency is not solely related to reducing the number of nodes but is also closely tied with other factors such as the cost of path backtracking (I3).

From Table II, our RL-based method demonstrates a notable ability to transfer effective strategies across different problem types. In the FCMCNF problem, our method significantly reduces both the solving time and the size of the B&B tree compared to all other methods. The RL method not only surpasses the SCIP strategies (ESTIMATE, DFS, and BFS), but it also outperforms the SVM, RankNet, and GNN models. This indicates that our RL method learned efficient general strategies from the training problems and was able to transfer this knowledge to the FCMCNF problem. In the WPMS problem, the RL method's solving time is not the best among all methods but is still competitive. Despite this, the RL method produces a significantly smaller B&B tree compared to RankNet and GNN, and a comparable size to the SVM method. These results underline the RL method's ability to transfer effective search strategies across different problem types. For the GISP problem, our RL method again demonstrates good performance, reducing the solving time compared to all methods except the DFS (SCIP). However, its B&B tree size is larger than GNN but smaller than SVM and RankNet, displaying a balanced performance.

Overall, the results from Table II highlight the strong transferability of the RL-based approach, allowing it to effectively

apply learned strategies to new problem types. This ability to generalize is critical for any ML-based approach aiming to enhance the efficiency of a MILP solver.

VI. Related work

Several machine learning approaches have been explored for solving MILP problems within branch-and-bound frameworks. He et al. [12] took the first strides towards leveraging machine learning for node comparison heuristics in branch-and-bound. They proposed training a support vector machine (SVM) to emulate the diving oracle's node comparison operator. Their work, however, was limited to conjunction with a learned pruning model, yielding a method more akin to a primal heuristic.

Subsequently, Song et al. [22] trained a multilayer perceptron RankNet model for node comparison, introducing retrospective imitation learning. They achieved significant improvements on specific path planning integer programs but reported less impactful results on more complex benchmarks.

Yilmaz and Yorke-Smith [13] proposed a limited form of feedforward neural network node comparison operator, combined with a backtracking algorithm, to provide a full node selection policy. They reported improvements in time and the number of nodes on several benchmarks.

However, these efforts have largely overlooked the cost of path backtracking and the significance of early stage tree exploration. For instance, on the WPMS dataset ($n \in [60, 70]$), the cost of path backtracking accounts for 15.2% of the total solving time. Our work, leveraging Reinforcement Learning (RL), addresses these aspects and has demonstrated performance that surpasses both SCIP's default methods and previous AI-based techniques.

VII. Conclusion

This work presents an innovative application of Reinforcement Learning (RL) in enhancing the efficiency of MILP solvers. By considering the cost of path backtracking and the significance of early search tree exploration, our approach offers a novel perspective to node selection strategy in the branch-and-bound algorithm. The superiority of our method is validated through comprehensive experimental comparisons, outperforming both SCIP's default strategies and existing AI/ML-based solutions.

Future work entails refining the strategy to further optimize the size of the branch-and-bound tree and integrate this RL-based node selection technique with other solver enhancements.

References

[1] V. T. Paschos, *Applications of combinatorial optimization*. John Wiley & Sons, 2014, vol. 3.

[2] C. A. Floudas and X. Lin, "Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications," *Annals of Operations Research*, vol. 139, pp. 131–162, 2005.

[3] M. K. Boujelben, C. Gicquel, and M. Minoux, "A milp model and heuristic approach for facility location under multiple operational constraints," *Computers & Industrial Engineering*, vol. 98, pp. 446–461, 2016.

[4] H. Ren and W. Gao, "A milp model for integrated plan and evaluation of distributed energy systems," *Applied energy*, vol. 87, no. 3, pp. 1001–1014, 2010.

[5] R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling, "Mixed-integer programming: A progress report," in *The sharpest cut: the impact of Manfred Padberg and his work*. SIAM, 2004, pp. 309–325.

[6] F. Friedler, J. Varga, E. Feher, and L. Fan, "Combinatorially accelerated branch-and-bound method for solving the mip model of process network synthesis," *State of the art in global optimization: computational methods and applications*, pp. 609–626, 1996.

[7] 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, and J. Witzig, "The SCIP Optimization Suite 8.0," Optimization Online, Technical Report, December 2021. [Online]. Available: http://www.optimization-online.org/DB_HTML/2021/12/8728.html

[8] G. O. Inc., "Gurobi optimizer." https://www.gurobi.com.

[9] I. B. M. Corporation, "Ibm cplex optimizer." https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer.

[10] K. Kianfar, "Branch-and-bound algorithms," *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[11] 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*, vol. 19, pp. 79–102, 2016.

[12] H. He, H. Daume III, and J. M. Eisner, "Learning to search in branch and bound algorithms," *Advances in neural information processing systems*, vol. 27, 2014.

[13] K. Yilmaz and N. Yorke-Smith, "Learning efficient search approximation in mixed integer branch and bound," *arXiv preprint arXiv:2007.03948*, 2020.

[14] A. G. Labassi, D. Chételat, and A. Lodi, "Learning to compare nodes in branch and bound with graph neural networks," *arXiv preprint arXiv:2210.16934*, 2022.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[16] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[18] 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*, vol. 34, pp. 24 235–24 246, 2021.

[19] C. Ansótegui and J. Gabàs, "Wpm3: an (in) complete algorithm for weighted partial maxsat," *Artificial Intelligence*, vol. 250, pp. 37–57, 2017.

[20] R. Béjar, A. Cabiscol, F. Manyà, and J. Planes, "Generating hard instances for maxsat," in *2009 39th International Symposium on Multiple-Valued Logic*. IEEE, 2009, pp. 191–195.

[21] M. Colombi, R. Mansini, and M. Savelsbergh, "The generalized independent set problem: Polyhedral analysis and solution approaches," *European Journal of Operational Research*, vol. 260, no. 1, pp. 41–55, 2017.

[22] J. Song, R. Lanka, A. Zhao, A. Bhatnagar, Y. Yue, and M. Ono, "Learning to search via retrospective imitation," *arXiv preprint arXiv:1804.00846*, 2018.