Planning in Branch-and-Bound: Model-Based Reinforcement Learning for Exact Combinatorial Optimization

Paul Strang[1, 2], **Zacharie Alès**[2,3], **Côme Bissuel**[1], **Olivier Juan**[1],
Safia Kedad-Sidhoum [2], **Emmanuel Rachelson** [4]
[1] EDF R&D
[2] CNAM
[3] ENSTA IP Paris
[4] ISAE-SUPAERO

Abstract

Mixed-Integer Linear Programming (MILP) lies at the core of many real-world combinatorial optimization (CO) problems, traditionally solved by branch-and-bound (B&B). A key driver influencing B&B solvers efficiency is the variable selection heuristic that guides branching decisions. Looking to move beyond static, hand-crafted heuristics, recent work has explored adapting traditional reinforcement learning (RL) algorithms to the B&B setting, aiming to learn branching strategies tailored to specific MILP distributions. In parallel, RL agents have achieved remarkable success in board games, a very specific type of combinatorial problems, by leveraging environment simulators to plan via Monte Carlo Tree Search (MCTS). Building on these developments, we introduce Plan-and-Branch-and-Bound (PlanB&B), a model-based reinforcement learning (MBRL) agent that leverages a learned internal model of the B&B dynamics to discover improved branching strategies. Computational experiments empirically validate our approach, with our MBRL branching agent outperforming previous state-of-the-art RL methods across four standard MILP benchmarks.

Code — https://github.com/abfariah/planbb
Extended version — https://arxiv.org/abs/2511.09219

1 Introduction

Mixed-Integer Linear Programming (MILP) plays a central role in combinatorial optimization (CO), a discipline concerned with finding optimal solutions over typically large but finite sets. Specifically, MILPs offer a general modeling framework for NP-hard problems, and have become indispensable in tackling complex decision-making tasks across fields as diverse as operations research (Hillier and Lieberman 2015), quantitative finance (Mansini et al. 2015) and computational biology (Gusfield 2019). Modern MILP solvers are built upon the branch-and-bound (B&B) paradigm (Land and Doig 1960), which systematically explores the solution space by recursively partitioning the original problem into smaller subproblems, while maintaining provable optimality guarantees. Since the 1980s, considerable research and engineering effort has gone into refining these solvers, resulting in highly optimized systems driven by expert-designed heuristics tuned over large benchmarks (Bixby 2012; Gleixner et al. 2021). Nevertheless, in operational settings where structurally similar problems are solved repeatedly, adapting solver heuristics to the distribution of encountered MILPs can lead to substantial gains in efficiency, beyond what static, hand-crafted heuristics can offer. Recent research has thus turned to machine learning (ML) to design efficient, data-driven B&B heuristics tailored to specific instance distributions (Scavuzzo et al. 2024). The variable selection heuristic, or branching heuristic, plays a particularly critical role in B&B overall computational efficiency (Achterberg and Wunderling 2013), as it governs the selection of variables along which the search space is recursively split. A key milestone was achieved by Gasse et al. (2019), who first managed to outperform human-expert branching heuristics by learning to replicate the behaviour of a greedy branching expert at lower computational cost. While subsequent works succeeded in learning efficient branching strategies by reinforcement (Etheve et al. 2020; Scavuzzo et al. 2022), none have yet matched the performance achieved by imitation learning (IL) approaches. This trend extends beyond MILPs to combinatorial optimization problems at large, as reinforcement learning (RL) baselines consistently underperfom both handcrafted heuristics and IL methods trained to replicate expert strategies across various CO benchmarks (Berto et al. 2023). Yet, if the performance of IL heuristics are caped by that of the experts they learn from, the performance of RL agents are, in theory, only bounded by the maximum score achievable. One of the main challenge lies in the high dimensionality and combinatorial complexity of CO problems, which exacerbates exploration and credit assignment issues in sequential decision making. While supervised learning can partially mitigate these issues by scaling up neural architectures and exploiting abundant labeled data, such approaches are impractical in RL due to sample inefficiency and unstable learning dynamics (Liu et al. 2023).

Despite these challenges, RL has achieved notable success in a specific subset of combinatorial problems, board games, reaching superhuman performance by leveraging environment simulators to perform model-based planning. Specifically, the integration of learned policy and value functions with look-ahead search via Monte Carlo Tree Search (MCTS) was found to steer action selection towards high-value states, effectively mitigating exploration and credit assignment is-

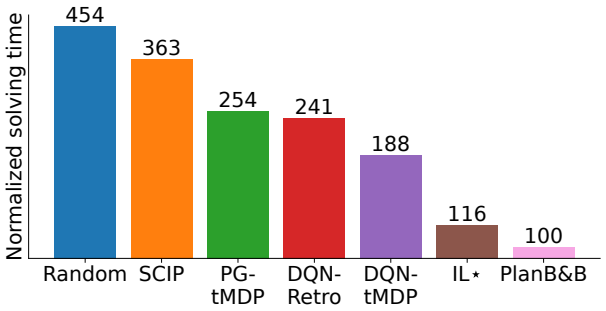


Figure 1: Aggregate normalized solving time performance obtained on test instances by SCIP (Bestuzheva et al. 2021), IL, RL and random baselines across the Ecole benchmark (Prouvost et al. 2020), in log scale. These baselines are formally introduced in Section 5.

sues in sparse-reward environment settings. Inspired by the work of Schrittwieser et al. (2020), we seek to extend the applicability of MCTS-based RL algorithms from board games to exact combinatorial optimization. To that end, we introduce Plan-and-Branch-and-Bound (PlanB&B), a model-based reinforcement learning (MBRL) agent that leverages an internal model of the B&B dynamics to learn improved variable selection strategies. To the best of our knowledge, this is the first MBRL agent specifically designed to solve CO problems. As shown in Figure 1, our agent achieves state-of-the-art performance on test sets across four standards MILP benchmarks (Prouvost et al. 2020), surpassing prior IL and RL baselines. These results suggest that the branching dynamics in B&B can be approximated with sufficient fidelity to enable policy improvement through planning over a learned model, opening the door to broader applications of MBRL to mixed-integer linear programming.

2 Problem Statement

Mixed-integer linear programming. We consider mixed-integer linear programs (MILPs), defined as:

$$P : \begin{cases} \min c^\top x \\ l \leq x \leq u \\ Ax \leq b \; ; \; x \in \mathbb{Z}^{|\mathcal{I}|} \times \mathbb{R}^{n-|\mathcal{I}|} \end{cases}$$

with n the number of variables, m the number of linear constraints, $l, u \in \mathbb{R}^n$ the lower and upper bound vectors, $A \in \mathbb{R}^{m \times n}$ the constraint matrix, $b \in \mathbb{R}^m$ the right-hand side vector, $c \in \mathbb{R}^n$ the objective function, and $\mathcal{I}$ the indices of integer variables. In this work, we are interested in repeated MILPs of fixed dimension $\{P_i = (A_i, b_i, c_i, l_i, u_i)\}_{i \in \mathcal{D}}$ sampled according to an unknown distribution p_0.

In order to solve MILPs efficiently, the B&B algorithm iteratively builds a binary tree $(\mathcal{V}, \mathcal{E})$ where each node corresponds to a MILP, starting from the root node $v_0 \in \mathcal{V}$ representing the original problem P_0. The incumbent solution $\bar{x} \in \mathbb{Z}^{|\mathcal{I}|} \times \mathbb{R}^{n-|\mathcal{I}|}$ denotes the best feasible solution found at current iteration, its associated value $GUB = c^\top \bar{x}$ is called the *global upper bound* on the optimal value. The overall state of the optimization process is thus captured by the triplet $s = (\mathcal{V}, \mathcal{E}, \bar{x})$, we note $\mathcal{S}$ the set of all such triplets. Throughout the optimization process, B&B nodes are explored sequentially. We note $\mathcal{C}$ the set of visited or closed nodes, and $\mathcal{O}$ the set of unvisited or open nodes, such that $\mathcal{V} = \mathcal{C} \cup \mathcal{O}$. Figure **??** illustrates how B&B operates on an example. Initially, $GUB = \infty$, $\mathcal{O} = \{v_0\}$, and $\mathcal{C} = \emptyset$. At each iteration $t \geq 0$, the node selection policy $\rho : \mathcal{S} \to \mathcal{O}$ selects the next node to explore. Let $x^*_{LP} \in \mathbb{R}^n$ be the optimal solution to the linear program (LP) relaxation of P_t, the MILP associated with the current node v_t:

- If the relaxation of P_t admits no solution, v_t is marked as closed and the branch is pruned by infeasibility. If $x^*_{LP} \in \mathbb{R}^n$ exists, and $GUB < c^\top x^*_{LP}$, no integer solution in P_t can improve GUB, thus v_t is marked as closed and the branch is pruned by bound. If x^*_{LP} is not dominated by $\bar{x}$ and x^*_{LP} is feasible, a new incumbent solution $\bar{x} = x^*_{LP}$ has been found. Hence, GUB is updated and v_t is marked as closed while the branch is pruned by integrity.
- Else, v_t is called *branchable*, as x^*_{LP} admits fractional values for some integer variables. The branching heuristic $\pi : \mathcal{S} \to \mathcal{I}$ selects a variable x_b with fractional value $\hat{x}_b$, to partition the solution space. As a result, two child nodes with associated MILPs $P_- = P_t \cup \{x_b \leq \lfloor \hat{x}_b \rfloor\}$ and $P_+ = P_t \cup \{x_b \geq \lceil \hat{x}_b \rceil\}$, are added to the set of open nodes $\mathcal{O}$ in place of v_t.[1]

This process is repeated until $\mathcal{O} = \emptyset$ and $\bar{x}$ is returned. The dynamics governing the B&B algorithm between two branching decisions can be described by the function $\kappa_\rho : \mathcal{S} \times \mathcal{I} \to \mathcal{S}$, such that $s' = \kappa_\rho(s, \pi(s))$. By design, B&B does not terminate before finding an optimal solution and proving its optimality. Consequently, optimizing the performance of B&B on a distribution of MILP instances is equivalent to minimizing the expected solving time of the algorithm. As Etheve (2021) evidenced, the variable selection strategy π is by far the most critical B&B heuristic in terms of computational performance. In practice, the total number of nodes of the B&B tree is used as an alternative metric to evaluate the performance of branching heuristics π, as it is a hardware-independent proxy for computational efficiency. Under these circumstances, given a fixed node selection strategy ρ, the optimal branching strategy π^* associated with a distribution p_0 of MILP instances can be defined as:

$$\pi^* = \arg\min_\pi \mathbb{E}_{P \sim p_0}(|BB_{(\pi,\rho)}(P)|) \tag{1}$$

with $|BB_{(\pi,\rho)}(P)|$ the size of the $B\&B$ tree after solving P to optimality following strategies (π, ρ).

Markov decision process formulation. The problem of finding an optimal branching strategy according to Eq. (1) can be described as a discrete-time deterministic Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, \mathcal{T}, p_0, \mathcal{R})$. State space $\mathcal{S}$ is the set of all B&B trees as defined previously, action space $\mathcal{A}$ is the set of all integer variables indices $\mathcal{I}$. Initial states are single node trees, where the root node is associated to a MILP P_0 drawn according to the distribution p_0. The Markov transition function is defined as $\mathcal{T} = \kappa_\rho$. Importantly, all states

[1] $\hat{x}_b$ denotes the value of x_b in x^*_{LP}. We use the symbol $\cup$ to denote the refinement of the bound on x_b in P_t.

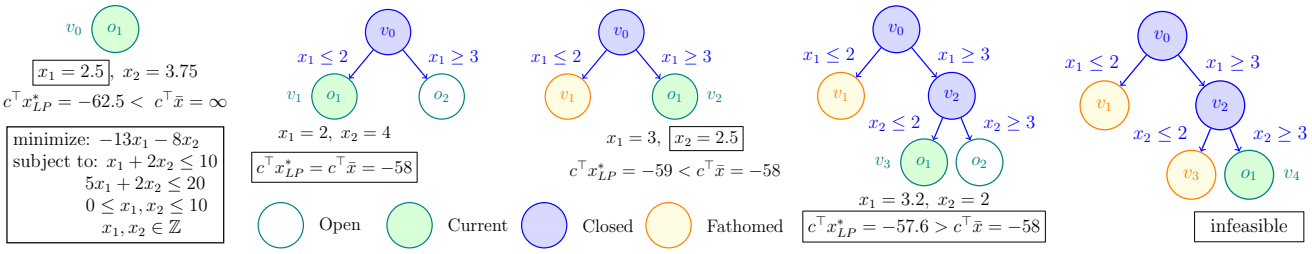


Figure 2: Solving a MILP by B&B using variable selection policy π and node selection policy ρ. Each node v_i represents a MILP derived from the original problem, each edge represents the bound adjustment applied to derive child nodes from their parent. Throughout the solving process, v_1 is pruned by integrity, v_3 is pruned by bound, and v_4 by infeasibility.

for which $\mathcal{O} = \emptyset$ are terminal states. The reward model is defined as $\mathcal{R}(s, a) = -1$ for all transitions until episode termination. In this setting, the Bellman state-value function writes $V^\pi(s) = -\mathbb{E}_{a' \sim \pi(s')}[\sum_{t' \geq 0} \gamma^{t'}]$ for $s \in \mathcal{S}$. Since episodes horizons are bounded by the (finite) largest possible number of nodes, we take a discount factor $\gamma = 1$. Indeed, under this formulation, the optimal policy π^* defined in Eq. (1) also maximizes the state-value function $V^\pi(s)$ for all $s \in \mathcal{S}$. Crucially, Etheve et al. (2020) and Scavuzzo et al. (2022) have shown that when the B&B tree is expanded following a depth-first-search (DFS) node selection policy, minimizing the total B&B tree size is achieved when any subtree is of minimal size. This allows the decomposition of B&B episodes into independent subtree trajectories, which helps mitigate credit assignment issues arising from the length of B&B episodes. Moreover, under $\rho = DFS$, V^π can be decomposed as the sum of the negative size $\bar{V}^\pi$ of the subtrees rooted in the open nodes of $s \in \mathcal{S}$:

$$V^\pi(s) = \sum_{o \in \mathcal{O}} \bar{V}^\pi(o, \bar{x}_o), \tag{2}$$

with $\bar{x}_o \in \mathbb{R}^n$ the incumbent solution when o is selected for expansion. Conveniently, this decomposition enables to derive V^π by training graph convolutionnal neural networks to approximate $\bar{V}^\pi$, using the MILP bipartite graph representation introduced by Gasse et al. (2019) as observation function for B&B nodes. Thus, as in previous works, we take $\rho = DFS$ in the remaining of the paper.

MuZero. Model-based reinforcement learning approaches exploit reversible access to the MDP to design policy improvement operators via planning (Silver et al. 2018; Hansen, Su, and Wang 2023). Monte Carlo Tree Search (MCTS) is among the most widely used planning algorithms in MBRL. Given access to an environment model $(\mathcal{T}, \mathcal{R})$, MCTS leverages value and policy estimates to select actions leading to promising states, while balancing an exploration-exploitation criterion. In order to extend the applicability of MCTS-based RL algorithms to broader control tasks where efficient simulators are not available, Schrittwieser et al. (2020) introduced MuZero, building upon the previous AlphaZero framework (Silver et al. 2018). Concretely, MuZero learns a model consisting of three interconnected networks. First, the representation network h maps raw state observations s_t to a latent state $\hat{s}_t = h(s_t)$. This internal state can then be passed to the prediction network f to obtain state policy and value estimates $(\hat{\mathrm{p}}_t, \hat{\mathrm{v}}_t)$. Alternatively, latent states can be passed along with an action a_t to the dynamics network g, to produce predictions for both the true reward r_t and the internal representation $h(s_{t+1})$ of the next visited state when taking action a_t at s_t: $g(\hat{s}_t, a_t) = (\hat{r}_t, \hat{s}_{t+1})$. At each decision step, MuZero integrates h, f and g to perform MCTS, thereby generating improved policy targets π_t from which actions are sampled. During training, the model is unrolled for K hypothetical steps and its predictions are aligned with sequences sampled from real trajectories collected by MCTS actors. By enforcing consistency between the predictions observed along simulated rollouts and those observed on real trajectories, MuZero constrains its model to capture only the information most relevant for accurately estimating future states' policy and value. This approach was shown to significantly reduce the burden of MDP dynamics approximation, unlocking the use of MBRL in visually complex domains where model-free RL methods had previously dominated. Subsequent work introduced several enhancements to the MuZero framework, including a temporal consistency loss for the dynamics network (Ye et al. 2021) and the use of Gumbel search for improved search efficiency in MCTS (Danihelka et al. 2022), which proved particularly useful in environments with large action space.

3 Planning in Branch-and-Bound

In the context of B&B, taking a step in the environment involves solving the two linear programs associated with the nodes generated by the branching decision. Unfortunately, this procedure is very expensive to simulate, and remains difficult to approximate accurately. In order to overcome these limitations, we introduce Plan-and-branch-and-bound (PlanB&B), a model-based reinforcement learning agent adapting the MuZero framework to the B&B setting. Crucially, our learned model is not explicitly trained to solve linear programs. Instead, it learns the dynamics of an abstract, value-equivalent MDP that retains only the aspects of B&B essential for enabling policy improvement via MCTS.

Model. Let $s_t = (\mathcal{V}_t, \mathcal{E}_t, \bar{x}_t) \in \mathcal{S}$ be a B&B tree where $\mathcal{V}_t = \mathcal{O}_t \cup \mathcal{C}_t$, and let $o = \rho(s_t) \in \mathcal{O}_t$ be the node currently selected for expansion. MuZero is originally designed

to operate on full state observations s_t as input to its internal model. In contrast, prior RL and IL branching agents have been designed to rely solely on information from the current B&B node, represented by the pair $(o, \bar{x}_t)$, and encoded using the MILP bipartite graph representation function from Gasse et al. (2019). While, under DFS, this local information is sufficient to recover the optimal policy at state s_t, it is generally insufficient to infer the subsequent state s_{t+1}. In fact, whenever taking action a_t at s_t leads to fathoming the subtree under o, the next visited node will be a leaf node that can not be deduced directly from the triplet $(o, \bar{x}_t, a_t)$. To address this, our model learns to predict recursively, **along subtree trajectories**, the internal representations $(\hat{o}_l, \hat{o}_r)$ associated with the left and right child nodes (o_l, o_r) generated by the agent's branching decisions.[2] Doing so, PlanB&B enables simulating imagined subtree trajectories while relying exclusively on the information available at the current node. The representation network h first maps the pair $(o, \bar{x}_t)$ to an internal representation $\hat{o}$ which serves as the root node for the imagined tree $\hat{\mathrm{T}} = (\hat{\mathcal{O}}, \hat{\mathcal{C}})$. Initially, $\hat{\mathcal{O}} = \{\hat{o}\}$ and $\hat{\mathcal{C}} = \emptyset$. The current imagined node $\hat{o}$ can then be passed to the dynamics network g along with any action in $\mathcal{A}$ to generate $(\hat{o}_l, \hat{o}_r)$. Since $r_t = -1$ is constant, g is not tasked with predicting the future reward. However, note that if either of the real child nodes is pruned, either by bound, integrity or infeasibility, its associated subtree value is null, and, consequently, its node internal representation should not be considered for expansion by the model. To distinguish nodes leading to branching decisions from nodes destined to be pruned, we task the prediction network f with estimating future nodes' *branchability* $\mathrm{b} \in \{0, 1\}$, in addition to policy and subtree value estimates[3]. Based on the predicted branchability values, PlanB&B updates the sets $(\hat{\mathcal{O}}, \hat{\mathcal{C}})$ by discarding unbranchable nodes and designates the next visited node as the node in $\hat{\mathcal{O}}$ with the greatest depth. If $\hat{\mathcal{O}} = \emptyset$, the imagined subtree has been fully explored, and all subsequent recursive calls to g will, by convention, receive a null reward. The overall interaction between the networks h, f, and g during the simulation of k-step subtree trajectories is illustrated in Figure 3. In-depth model description as well as network architectures for h, f and g are presented in Appendix B.

Data generation. Since $\rho = DFS$, learning a policy minimizing the local subtree size is equivalent to learning a globally optimal policy. Therefore, we can use the model introduced in the previous paragraph to derive improved branching policy targets π_t through planning. Due to the large action space size typically encountered in MILP environments, we implement Gumbel search (Danihelka et al. 2022), a variant of MCTS presented in Appendix C. Throughout the search, the policy prior $\hat{\mathrm{p}}_t^k$ associated with the imagined tree $\hat{\mathrm{T}}^k$ is given by the policy prediction of the branchable node in $\hat{\mathcal{O}}^k$ with the greatest depth. Similarly, following Eq. (2), the

[2]Following our convention, in DFS, o_l designates the node selected immediately next for expansion, while o_r designates the node selected once the subtree rooted in o_l has been fathomed.

[3]By construction, κ_ρ ensures that the current node at s_t is always branchable.

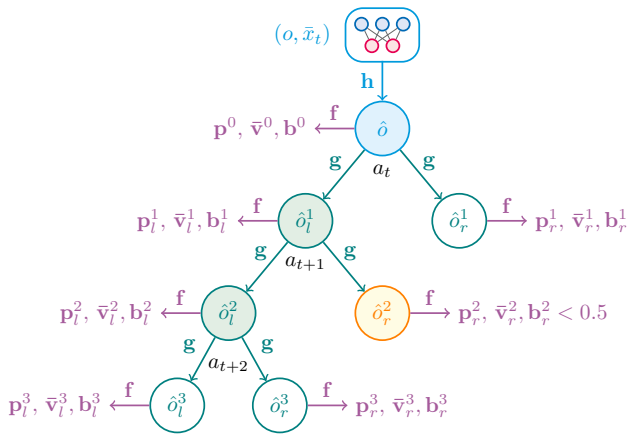


Figure 3: Planning in B&B over a learned model. The combined use of h, f, and g allows simulating subtree rollouts starting from the current B&B node. Here, $\hat{\mathrm{T}}^3 = (\hat{\mathcal{O}}^3, \hat{\mathcal{C}}^3)$ with $\hat{\mathcal{O}}^3 = \{\hat{o}_l^3, \hat{o}_r^3, \hat{o}_r^1\}$ and $\hat{\mathcal{C}}^3 = \{\hat{o}_r^2\}$. To simplify notations, we write z_j^i in place of $\mathrm{z}_{\hat{o}_j^i}$ for $\mathrm{z} \in \{\mathrm{p}, \bar{\mathrm{v}}, \mathrm{b}\}$.

estimate state value $\hat{\mathrm{v}}_t^k$ is obtained by summing the subtree value predictions $\bar{\mathrm{v}}_{\hat{o}}$ of all open nodes in the imagined tree: $\hat{\mathrm{v}}_t^k = \sum_{\hat{o} \in \hat{\mathcal{O}}^k} \bar{\mathrm{v}}_{\hat{o}}$. Thus, by combining networks h, f and g to simulate subtree rollouts, Gumbel search yields an improved target policy π_t from which the selected action a_t is sampled. Repeating this process generates B&B episodes that are stored in memory and subsequently used for training.

Learning. The agent is trained over K-step subtree trajectories $(s_t, a_t, \ldots, s_{t+K})$ sampled from memory. As in MuZero, the model is unrolled from s_t over K steps to generate imagined trajectories $(\hat{\mathrm{T}}^0, \ldots, \hat{\mathrm{T}}^K)$. As illustrated in Figure 3, for each step $k = 0, \ldots, K$, the model recursively predicts $\hat{\mathrm{p}}_t^k$ and $\hat{\mathrm{v}}_t^k$ to approximate the Gumbel search policy π_{t+k} and n-step return $z_{t+k} = -n + \hat{\mathrm{v}}_t^{k+n}$ prediction targets. Moreover, the predicted branchability scores of imagined nodes are trained to match the true branchability labels of real nodes. To support this, we introduce a new self-supervised loss term $\mathcal{L}_{\mathrm{T}}$, which enforces structural consistency between real and imagined subtrees, as well as hierarchical consistency between consecutive B&B node observations. The overall loss minimization objective can thus be summarized as:

$$\mathcal{L}_t(\theta) = \sum_{k=0}^{K} \mathcal{L}_{\mathrm{p}}(\pi_{t+k}, \hat{\mathrm{p}}_t^k) + \mathcal{L}_{\mathrm{v}}(z_{t+k}, \hat{\mathrm{v}}_t^k) + \mathcal{L}_{\mathrm{T}}(s_{t+K}, \hat{\mathrm{T}}_K)$$

with θ the global parameter for f, g, h. Losses $\mathcal{L}_{\mathrm{p}}$, $\mathcal{L}_{\mathrm{v}}$ and $\mathcal{L}_{\mathrm{T}}$ are presented in Appendix E.

4 Related Work

Following the work of Gasse et al. (2019), several studies have explored adopting more expressive neural architectures, from transformer-based models (Lin et al. 2022) to recurrent designs (Seyfi et al. 2023), for the purpose of learning to branch. However, these architectural extensions have yielded

only limited gains. In parallel, theoretical and empirical analyses (Dey et al. 2021) have shown that the small trees produced by strong branching (SB), the branching expert used in Gasse et al. (2019), arise mainly from the extensive formulation tightening induced by solving large numbers of LPs, rather than from the intrinsic quality of its branching decisions.

Such findings cast doubt on the long-term effectiveness of purely imitation-based approaches. Since branching decisions unfold sequentially, reinforcement learning provides a principled alternative framework for learning to branch. Building on the works of Etheve et al. (2020) and Scavuzzo et al. (2022), Parsonson et al. (2022) explored training RL branching agents on B&B trees expanded following solvers' default node selection policies, rather than DFS. To address the partial observability induced by moving away from DFS, they trained their agent on retrospective trajectories, tree-diving trajectories extracted from original B&B episodes.

More generally, a substantial line of research (Nair et al. 2021; Paulus et al. 2022) has explored augmenting B&B heuristics with machine learning algorithms beyond variable selection. Building on the MDP formulation first proposed by He, Daume III, and Eisner (2014), recent RL contributions to primal search (Sonnerat et al. 2022; Wu and Lisser 2023), node selection (Etheve 2021; Zhang et al. 2025), and cut selection (Tang, Agrawal, and Faenza 2020; Wang et al. 2023) have primarily focused on deploying traditional model-free RL algorithms within the B&B framework, adapting the action space to suit the specific heuristic being targeted.

Finally, machine learning methods have been applied beyond the scope of exact combinatorial optimization. In large routing and scheduling problems, where exact resolution quickly becomes intractable, recent works have trained neural agents to act as direct-search heuristics capable of producing strong feasible solutions (Kool, van Hoof, and Welling 2019; Chalumeau et al. 2023). Drawing inspiration from model-based planning, Pirnay and Grimm (2024) proposed a self-improvement operator that refines learning agents' policies by generating alternative trajectories via stochastic beam search, and treating improved trajectories as targets for imitation learning. Applied to routing problems, their approach matched the performance of imitation learning agents trained on expert demonstrations, highlighting the potential of planning-based methods as a data-efficient alternative to expert supervision in combinatorial optimization.

5 Experimental Study

We now assess the efficiency of our model-based branching agent, as we aim to answer the following questions:

Q1 Can PlanB&B learn an efficient policy network to guide MILP solving? In particular, how does it compare against solver heuristics, as well as prior RL and IL approaches?

Q2 When provided with additional search budget, can our agent further improve the quality of its decisions by leveraging its internal model of B&B?

Q3 To what extent does the branching behavior of PlanB&B align with that of the expert strong branching (SB) strategy?

Q4 Is the use of a DFS node selection policy inherently detrimental to the performance of branching agents?

Benchmarks. We consider four standard MILP benchmarks: set covering (SC), combinatorial auctions (CA), maximum independent set (MIS) and multiple knapsack (MK) problems. In our experiments, SCIP 8.0.3 (Bestuzheva et al. 2021) is used as backend MILP solver, along with the Ecole library (Prouvost et al. 2020) for instance generation, see Appendix A for further benchmark details.

Baselines. We compare our PlanB&B agent against prior RL agents, namely DQN-tMDP (Etheve et al. 2020), PG-tMDP (Scavuzzo et al. 2022) and DQN-Retro (Parsonson et al. 2022). We also compare against the IL expert from Gasse et al. (2019), evaluated under both SCIP's default node selection policy (IL*), and DFS (IL-DFS). More details on these baselines can be found in Appendix G. Finally, we report the performance of reliability pseudo cost branching (SCIP), the default branching heuristic used in SCIP, strong branching (SB) (Applegate et al. 1995), the greedy expert from which the IL agent learns from, and random branching (Random), which randomly selects a fractional variable. SCIP configuration is common to all baselines. As in prior works, we set the time limit to one hour, disable restart, and deactivate cut generation beyond root node. All the other parameters are left at their default value.

Training & evaluation. The overall PlanB&B training pipeline is provided in Appendix F. Branching agents are trained on instances of each benchmark separately. For evaluation, we report performance in terms of both node count and solving time on 100 test instances unseen during training, as well as on 100 transfer instances of higher dimensions. Evaluation metrics are averaged over 5 random seeds. Importantly, when comparing ML-based branching strategies to standard SCIP heuristics such as RPB or SB, solving time remains the only reliable performance indicator. This is because invoking a custom branching rule in SCIP triggers auxiliary routines, such as conflict analysis and bound tightening, that strengthen the MILP formulation while incurring computational overhead.

Baselines comparison (Q1) Computational results obtained on the four benchmarks are presented in Table 1. Standard deviations, additional performance metrics as well as targeted ablations are provided in Appendix I. Unlike conventional MBRL settings, PlanB&B operates under strict time and computational constraints. Crucially, every second spent on planning directly increases the overall solving time, thereby impacting final performance. Therefore, to allow systematic comparison, the results reported for PlanB&B in Table 1 reflect the performance of its policy network alone, without any computational budget allocated to MCTS simulations at evaluation time. On aggregate test instances, compared to prior RL baselines, PlanB&B's policy network achieves $2\times$ reductions both in tree size and solving time. This performance gap broadens further to $3\times$ on aggregate transfer instances, underscoring the superior generalization capabilities of PlanB&B relative to prior RL agents. PlanB&B also outperforms the IL-DFS agent on both test and transfer instances, providing, to our knowledge, the first evidence of an RL-based branching strategy surpassing

Method	Set Covering Node	Set Covering Time	Comb. Auction Node	Comb. Auction Time	Max. Ind. Set Node	Max. Ind. Set Time	Mult. Knapsack Node	Mult. Knapsack Time	Norm. Score ± Conf. Interval Node	CI	Time	CI
Presolve	–	4.74	–	0.90	–	1.78	–	0.20	–	–	–	–
Random	3289	5.94	1111	2.16	386.8	2.01	733.5	0.55	1068	±113	454	±51
SB	35.8	12.93	28.2	6.21	24.9	45.87	161.7	0.69	46	±0	4279	±57
SCIP	62.0	2.27	20.2	1.77	19.5	2.44	289.5	0.53	58	±0	363	±3
IL*	**133.8**	0.90	**83.6**	0.65	**40.1**	0.36	272.0	0.69	**96**	**±17**	116	±15
IL-DFS	136.4	**0.74**	92.1	0.56	68.5	0.44	411.5	1.07	130	±26	131	±20
PG-tMDP	649.4	2.32	168.0	0.94	153.6	0.92	436.9	1.57	272	±46	254	±50
DQN-tMDP	**175.8**	**0.83**	203.3	1.11	168.0	1.00	266.4	0.73	207	±25	188	±16
DQN-Retro	183.0	1.14	103.2	0.78	223.0	1.81	250.3	0.67	208	±23	241	±25
PlanB&B	186.2	0.87	**84.7**	**0.54**	**44.8**	**0.32**	**220.0**	**0.55**	**100**	**±9**	**100**	**±12**

Test instances

Method	Set Covering Node	Set Covering Time	Comb. Auction Node	Comb. Auction Time	Max. Ind. Set Node	Max. Ind. Set Time	Mult. Knapsack Node	Mult. Knapsack Time	Norm. Score ± Conf. Interval Node	CI	Time	CI
Presolve	–	12.3	–	2.67	–	5.16	–	0.46	–	–	–	–
Random	271632	842	317235	749	215879	2102	93452	70.6	8050	±1646	3870	±847
SB	672.1	398	389.6	255	169.9	2172	**1709**	**12.5**	13	±0	2243	±47
SCIP	3309	48.4	1376	14.77	3368	90.0	30620	22.1	121	±0	132	±1
IL*	**2610**	23.1	**1282**	9.4	**1993.0**	**38.6**	11730	43.5	**70**	**±7**	**83**	**±7**
IL-DFS	3103	**22.5**	1828	10.2	3348	51.9	43705	130.8	151	±30	136	±21
PG-tMDP	44649	221	6001	30.7	3133	43.6	35614	123	373	±48	290	±42
DQN-tMDP	8632	71.3	20553	116	45634	477	22631	65.1	787	±128	679	±78
DQN-Retro	6100	59.4	2908	18.4	119478	1863	27077	79.5	1166	±256	1254	±218
PlanB&B	**5869**	**46.2**	**1665**	**9.1**	**2853**	**41.1**	**13574**	**51.2**	**100**	**±9**	**100**	**±8**

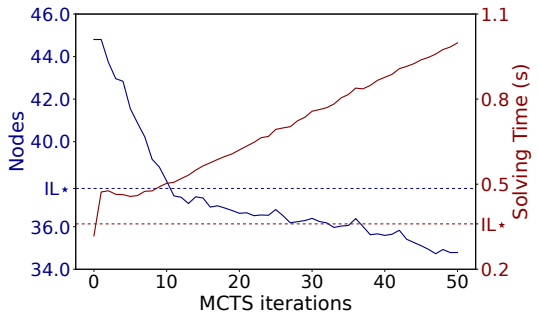
Transfer instances

Table 1: Performance comparison of branching agents on four standard MILP benchmarks. For each method, we report the total number of B&B nodes, presolve time, and total solving time excluding presolve. The presolve phase is identical across all methods. Lower values indicate better performance. **Red** highlights the overall best agent, while **blue** marks the best-performing RL-based agent. Following prior work, results are reported as the geometric mean over 100 unseen test instances and an additional 100 higher-dimensional transfer instances, averaged across 5 random seeds. Norm. Score represents the aggregate average performance of each agent across the four MILP benchmarks, normalized by the score of PlanB&B. Aggregate confidence intervals (CI) are reported for each baselines; additional per-benchmark confidence intervals are provided in Appendix I.

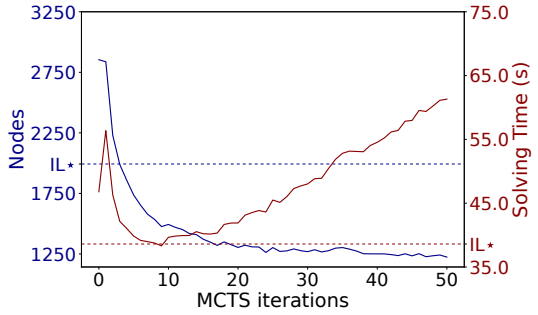
an IL agent trained to mimic strong branching. Specifically, our experiments demonstrate that, under a DFS node selection policy, PlanB&B learns branching strategies superior to that of the IL agent from Gasse et al. (2019). When compared against the standard IL* baseline, PlanB&B manages to achieve $\approx 10\%$ lower solving time on test instances, despite producing $\approx 5\%$ larger trees in average. However, on transfer instances, PlanB&B is globally unable to overcome the performance limitations imposed by DFS, although it still manages to outperform the IL* baseline on combinatorial auction. The influence of DFS on the performance of branching agents is further analyzed in **(Q4)**. Finally, PlanB&B clearly outperforms the default SCIP baseline on both test and transfer instances, despite operating under DFS, a performance achieved by no other learning-based baseline.

Planning in B&B (Q2) Although PlanB&B demonstrates strong performance while relying solely on its policy network, we further investigate its capacity to derive stronger policies by leveraging its learned model to plan at evaluation time. Figure 4 shows the effect of increasing PlanB&B's simulation budget N on both node and time performance over MIS test and transfer instances. On test instances, PlanB&B achieves lower average tree size than the IL* baseline as soon as $N > 12$, reaching up to $\approx 20\%$ tree size reduction for $N = 50$. On transfer instances, not only increased simulation budget enables to reduce average tree size up to $\approx 50\%$, but this reduction also translates into improved solving time performance, with best solving time, achieved at $N = 9$, matching the solving time performance of the IL* agent. Remarkably, when planning over its internal model, PlanB&B produces branching strategy yielding smaller trees than that produced by the IL* agent, despite operating under DFS.

Is PlanB&B learning to strong branch? (Q3) Given the performance trends observed in Figure 4, a natural question arises: does PlanB&B manage to derive better branching decisions than IL baselines by following SB more closely, or does it discover genuinely novel strategies? After all, strong branching can be interpreted as a one-step MCTS procedure



(a) Test instances



(b) Transfer instances

Figure 4: Policy improvement associated with increased simulation budget over the MIS benchmark.

aimed at maximizing immediate dual gap reduction. To answer this, Table 2 reports several alignment metrics designed to assess which of the IL* and PlanB&B policies more closely resembles strong branching. Across all metrics, PlanB&B policies exhibit lower alignment with SB than IL* policies. Remarkably, the refined branching policy returned by the MCTS is roughly as close to SB as the policy yielded by the policy network. This suggests that PlanB&B surpasses IL baselines not through closer imitation of strong branching, but by discovering and exploiting original strategies.

Influence of DFS (Q4) We finally turn to analyzing the impact of DFS on the performance of branching agents. From Table 1, two key observations emerge. First, the performance gap between DFS and non-DFS variants varies significantly across benchmarks: for instance, MK exhibits a particularly large gap, while SC shows a much narrower one. Second, across all benchmarks, this gap consistently widens as the problem dimensionality increases from test to transfer instances. This trend can be attributed to the typical *discovery step* t_d at which the B&B algorithm discovers the global optimal solution x^*. In fact, as soon as $\bar{x} = x^*$, the primal gap is closed, and, consequently, all node selection policies are equivalent for $t \geq t_d$. Therefore, in theory, the smaller the value of t_d, the smaller the performance gap. Table 3 reports both absolute and relative average discovery times obtained by IL on SC and MK benchmarks. The results are consistent with our theoretical intuition, see Appendix I for further discussion. Moreover, they highlight a fundamental

Instances	Policy	Iter.	(↓) SB C-Entropy	(↑) SB Score	(↑) SB Freq.
	SB	–	0.00	1.00	1.00
Test	IL*	–	0.84	0.69	0.45
	PlanB&B	0	1.97	0.63	0.39
	PlanB&B	50	2.08	0.65	0.40
Transfer	IL*	–	0.86	0.76	0.40
	PlanB&B	0	1.46	0.72	0.39
	PlanB&B	50	1.34	0.71	0.38

Table 2: Alignment metrics between ML baselines and SB on MIS test and transfer instances. Obviously, SB is perfectly aligned with SB. Experimental setup and thorough metrics description is provided in Appendix I.

	SC Test	SC Transfer	MK Test	MK Transfer
t_d	29	514	376	10934
t_r	0.14	0.25	1.0	1.0

Table 3: IL* agent discovery times, with $t_r = t_d/T$ on SC and MIS benchmarks, see Appendix I for further results.

limitation of applying DFS when solving higher-dimensional MILPs: as problem size grows, closing the primal gap becomes increasingly difficult, which in turn exacerbates the computational burden associated with DFS. To address this issue, future RL contributions need moving beyond the traditional MILP bipartite graph representation and adopt more expressive global representations of the B&B tree, as recently proposed by Zhang et al. (2025).

6 Conclusion & Perspectives

Combinatorial optimization has proven in the past to be a challenging setting for traditional RL approaches. In this work, we introduced PlanB&B, a novel model-based reinforcement learning framework leveraging a learned internal model of B&B to discover new variable selection strategies. Our experimental study leads to three main findings. First, PlanB&B's dynamics network approximates LP resolution in the latent space with sufficient fidelity to enable policy improvement through model-based planning, thereby extending the applicability of MBRL algorithms originally developed for combinatorial board games to mixed-integer linear programming. Second, in the context of repeated MILPs, our MBRL agent learns branching strategies that clearly outperform both SCIP and former RL baselines. Moreover, further analysis shows instances where PlanB&B surpasses IL, not by more closely replicating expert behavior, but by actively diverging from strong branching patterns, highlighting the potential of RL to uncover branching strategies going beyond existing expert heuristics. Third and finally, our computational study highlights the burden imposed by the DFS node selection policy when seeking to solve higher-dimensional MILPs. In order to fully unlock the potential of MBRL in exact combinatorial optimization, we expect future research to explore the design of scalable observation functions capable of efficiently encoding evolving B&B trees.

References

Achterberg, T.; and Wunderling, R. 2013. Mixed integer programming: Analyzing 12 years of progress. In *Facets of combinatorial optimization: Festschrift for martin grötschel*, 449–481. Springer.

Applegate, D.; Bixby, R.; Chvátal, V.; and Cook, W. 1995. *Finding cuts in the TSP (A preliminary report)*, volume 95. Citeseer.

Berto, F.; Hua, C.; Park, J.; Kim, M.; Kim, H.; Son, J.; Kim, H.; Kim, J.; and Park, J. 2023. RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark. ArXiv:2306.17100 [cs].

Bestuzheva, K.; Besançon, M.; Chen, W.-K.; Chmiela, A.; Donkiewicz, T.; Doornmalen, J. v.; Eifler, L.; Gaul, O.; Gamrath, G.; Gleixner, A.; Gottwald, L.; Graczyk, C.; Halbig, K.; Hoen, A.; Hojny, C.; Hulst, R. v. d.; Koch, T.; Lübbecke, M.; Maher, S. J.; Matter, F.; Mühmer, E.; Müller, B.; Pfetsch, M. E.; Rehfeldt, D.; Schlein, S.; Schlösser, F.; Serrano, F.; Shinano, Y.; Sofranac, B.; Turner, M.; Vigerske, S.; Wegscheider, F.; Wellner, P.; Weninger, D.; and Witzig, J. 2021. The SCIP Optimization Suite 8.0. Technical Report, Optimization Online.

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

Chalumeau, F.; Surana, S.; Bonnet, C.; Grinsztajn, N.; Pretorius, A.; Laterre, A.; and Barrett, T. 2023. Combinatorial optimization with policy adaptation using latent space search. *Advances in Neural Information Processing Systems*, 36: 7947–7959.

Danihelka, I.; Guez, A.; Schrittwieser, J.; and Silver, D. 2022. Policy improvement by planning with Gumbel. In *International Conference on Learning Representations*.

Dey, S. S.; Dubey, Y.; Molinaro, M.; and Shah, P. 2021. A Theoretical and Computational Analysis of Full Strong-Branching. ArXiv:2110.10754 [math].

Etheve, M. 2021. *Solving repeated optimization problems by Machine Learning*. phdthesis, HESAM Université.

Etheve, M.; Alès, Z.; Bissuel, C.; Juan, O.; and Kedad-Sidhoum, S. 2020. Reinforcement Learning for Variable Selection in a Branch and Bound Algorithm. In Hebrard, E.; and Musliu, N., eds., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Lecture Notes in Computer Science, 176–185. Cham: Springer International Publishing. ISBN 978-3-030-58942-4.

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

Gleixner, A.; Hendel, G.; Gamrath, G.; Achterberg, T.; Bastubbe, M.; Berthold, T.; Christophel, P.; Jarck, K.; Koch, T.; and Linderoth, J. 2021. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3): 443–490. Publisher: Springer.

Gusfield, D. 2019. *Integer linear programming in computational and systems biology: an entry-level text and course*. Cambridge University Press.

Hansen, N.; Su, H.; and Wang, X. 2023. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*.

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

Hillier, F. S.; and Lieberman, G. J. 2015. *Introduction to operations research*. McGraw-Hill.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, Learn to Solve Routing Problems! ArXiv:1803.08475 [cs, stat].

Land, A.; and Doig, A. 1960. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28(3): 497–520.

Lin, J.; Zhu, J.; Wang, H.; and Zhang, T. 2022. Learning to branch with Tree-aware Branching Transformers. *Knowledge-Based Systems*, 252: 109455.

Liu, H.; Kuang, Y.; Wang, J.; Li, X.; Zhang, Y.; and Wu, F. 2023. Promoting Generalization for Exact Solvers via Adversarial Instance Augmentation. ArXiv:2310.14161 [cs].

Mansini, R.; Ogryczak, W.; Speranza, M. G.; and of European Operational Research Societies, E. T. A. 2015. *Linear and mixed integer programming for portfolio optimization*, volume 21. Springer.

Nair, V.; Bartunov, S.; Gimeno, F.; von Glehn, I.; Lichocki, P.; Lobov, I.; O'Donoghue, B.; Sonnerat, N.; Tjandraatmadja, C.; Wang, P.; Addanki, R.; Hapuarachchi, T.; Keck, T.; Keeling, J.; Kohli, P.; Ktena, I.; Li, Y.; Vinyals, O.; and Zwols, Y. 2021. Solving Mixed Integer Programs Using Neural Networks. ArXiv:2012.13349 [cs, math].

Parsonson, C. W. F.; Laterre, A.; Barrett, T. D.; Doe, J.; and Doe, J. 2022. Reinforcement Learning for Branch-and-Bound Optimisation using Retrospective Trajectories. ArXiv:2205.14345 [cs].

Paulus, M. B.; Zarpellon, G.; Krause, A.; Charlin, L.; and Maddison, C. 2022. Learning to Cut by Looking Ahead: Cutting Plane Selection via Imitation Learning. In *Proceedings of the 39th International Conference on Machine Learning*, 17584–17600. PMLR. ISSN: 2640-3498.

Pirnay, J.; and Grimm, D. G. 2024. Self-improvement for neural combinatorial optimization: Sample without replacement, but improvement. *arXiv preprint arXiv:2403.15180*.

Prouvost, A.; Dumouchelle, J.; Scavuzzo, L.; Gasse, M.; Chételat, D.; and Lodi, A. 2020. Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers. ArXiv:2011.06069 [cs, math].

Scavuzzo, L.; Aardal, K.; Lodi, A.; and Yorke-Smith, N. 2024. Machine Learning Augmented Branch and Bound for Mixed Integer Linear Programming. ArXiv:2402.05501 [cs, math].

Scavuzzo, L.; Chen, F. Y.; Chételat, D.; Gasse, M.; Lodi, A.; Yorke-Smith, N.; and Aardal, K. 2022. Learning to branch with Tree MDPs. ArXiv:2205.11107 [cs, math].

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; Lillicrap, T.; and Silver, D. 2020. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609. Number: 7839 Publisher: Nature Publishing Group.

Seyfi, M.; Banitalebi-Dehkordi, A.; Zhou, Z.; and Zhang, Y. 2023. Exact Combinatorial Optimization with Temporo-Attentional Graph Neural Networks. ArXiv:2311.13843 [cs].

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Sonnerat, N.; Wang, P.; Ktena, I.; Bartunov, S.; and Nair, V. 2022. Learning a Large Neighborhood Search Algorithm for Mixed Integer Programs. ArXiv:2107.10201 [cs, math].

Tang, Y.; Agrawal, S.; and Faenza, Y. 2020. Reinforcement Learning for Integer Programming: Learning to Cut. In *Proceedings of the 37th International Conference on Machine Learning*, 9367–9376. PMLR. ISSN: 2640-3498.

Wang, Z.; Li, X.; Wang, J.; Kuang, Y.; Yuan, M.; Zeng, J.; Zhang, Y.; and Wu, F. 2023. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. *arXiv preprint arXiv:2302.00244*.

Wu, D.; and Lisser, A. 2023. A deep learning approach for solving linear programming problems. *Neurocomputing*, 520: 15–24.

Ye, W.; Liu, S.; Kurutach, T.; Abbeel, P.; and Gao, Y. 2021. Mastering Atari Games with Limited Data. ArXiv:2111.00210 [cs].

Zhang, S.; Zeng, S.; Li, S.; Wu, F.; and Li, X. 2025. Learning to Select Nodes in Branch and Bound with Sufficient Tree Representation. In *The Thirteenth International Conference on Learning Representations*.