Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

SOLVING MIXED INTEGER LINEAR PROGRAMS BY MONTE CARLO TREE SEARCH

Gongbo Zhang[1,2,3], and Yijie Peng[1,2,3]

[1]PKU-Wuhan Institute for Artificial Intelligence, Peking University, Wuhan, CHINA
[2]Guanghua School of Management, Peking University, Beijing, CHINA
[3]Xiangjiang Laboratory, Changsha, CHINA

ABSTRACT

Mixed Integer Linear Programs (MILPs) are powerful tools for modeling and solving combinatorial optimization problems. Solving an MILP is NP-hard due to the integrality requirement, and the branch and bound (B&B) algorithm is a widely used exact solution method. In this work, we explore the use of Monte Carlo Tree Search (MCTS) to guide the search within the space composed of branching candidate variables, aiming to efficiently find the optimal solution (if it exists) for an MILP. We adapt the Asymptotically Optimal Allocation for Trees (AOAT) algorithm, a recently proposed MCTS approach in the simulation and optimization field, for solving MILPs. Numerical results demonstrate the potential benefits of the proposed method.

1 INTRODUCTION

Mixed Integer Linear Programs (MILPs) are a type of optimization problem where the objective is to find the optimal solution while meeting specific linear constraints, with some variables required to be integers. MILPs are commonly used in scheduling, routing, resource allocation, and various other combinatorial optimization problems (Wolsey and Nemhauser 2014). Typically, MILPs are solved using a variant of the branch and bound (B&B) (Land and Doig 1960) algorithm. The B&B algorithm is a tree-based method that iteratively partitions fractional variables that need to be integers and computes relaxation bounds to prune subtrees. The tree search structure in the B&B algorithm allows for the adoption of Monte Carlo Tree Search (MCTS) (Fu 2018; Świechowski et al. 2023) to solve MILPs. In this work, we focus on a recently proposed MCTS algorithm, referred to as Asymptotically Optimal Allocation for Tree (AOAT) (Zhang et al. 2022; Liu et al. 2023), which extends the Asymptotically Optimal Allocation Policy (AOAP) (Peng et al. 2018; Zhang et al. 2023), a sampling policy originally developed for the ranking and selection (R&S) problem, to searching within tree-structured spaces. Following the scheme of the B&B algorithm, we extend AOAT to solve MILPs, allowing AOAT to guide the search within the space composed of branching candidate variables toward promising areas, thus increasing the likelihood of quickly finding the primal optimal solution (if it exists) for an MILP.

The B&B algorithm is widely used to produce exact solutions for non-convex and combinatorial problems that cannot be solved in polynomial time. See Linderoth and Savelsbergh (1999) and Huang et al. (2021) for overviews of B&B strategies. Several modern MILP solvers, such as CPLEX and Gurobi, use B&B strategies to solve MILP instances. An MILP is challenging to solve because integer variables require efficient handling. However, a linear programming (LP) relaxation of the corresponding MILP, by removing the integrality constraints, can be more easily solved using algorithms like the simplex method (Dantzig 1951). The B&B algorithm is a tree-based strategy that starts with an LP relaxation and uses a divide-and-conquer approach by iteratively partitioning the search space, computing relaxation bounds to prune subtrees that provably cannot contain an optimal solution. The exact optimal solution can be found once the entire search tree has been explored. In the B&B strategy, two key decisions are made at each iteration: how to split a problem (branching) and which subproblem to select next (fixing the branching

979-8-3315-3420-2/24/$31.00 ©2024

variable to either its floor or ceiling value). The B&B algorithm can be slow, depending on the selection of branching rules, and the computational cost may grow exponentially with the size of the MILP. There have been several attempts to develop effective branching strategies (Achterberg et al. 2005), including the most infeasible branching, pseudocost branching (Bénichou et al. 1971; Linderoth and Savelsbergh 1999), strong branching (Applegate, David and Bixby, Robert 1995), and graph convolutional network branching (Gasse et al. 2019), which can reduce the search space and speed up the B&B algorithm significantly. Our work focuses on efficiently exploring a search tree to solve MILPs, rather than developing efficient branching strategies within the B&B algorithm.

Strategies like depth-first search (exploring deep into the tree before backtracking) (Dakin 1965) and best-first search (exploring nodes with the most promising bounds) (Achterberg 2007) have been used to guide the search in the B&B tree. Given that LPs are typically solved relatively quickly, diving heuristics have been developed to efficiently guide the LP of the original problem toward integrality to find feasible solutions in the B&B tree, conducting a depth-first search by iteratively modifying and solving LPs (Paulus and Krause 2024). The tree-search strategies in B&B motivate this work to utilize MCTS to solve MILPs, where MCTS balances between depth-first and best-first search, allowing exploration of a wide range of possible solutions while still focusing on the most promising ones. MCTS is a popular tree-based search strategy within the framework of reinforcement learning (RL), which combines the structure of a search tree with the randomness and exploration of Monte Carlo simulations. This approach aids in making informed decisions in complex decision-making environments, especially those with large state and action spaces. MCTS has been highly successful in various applications, notably in designing expert computer players for challenging games such as Go (Enzenberger et al. 2010; Silver et al. 2017), and in large-scale combinatorial optimization problems (Świechowski et al. 2023). However, games like Go differ qualitatively from MILPs. MCTS for Go aims to find the best move by estimating the state value of child nodes from the root node, whereas MCTS for MILPs aims to guide the tree search toward optimal areas to find the best solutions, with the objective value of the LP at the leaf node providing meaningful information.

The decision-making process in the B&B strategy has been formulated as a Markov Decision Process (MDP) (Gasse et al. 2019; Mazyavkina et al. 2021), providing an opportunity to combine RL algorithms with the B&B algorithm. Tang et al. (2020) propose an attention network for making cutting plane decisions for a specific feasible region. Deep learning methods with neural networks (Gasse et al. 2019; Nair et al. 2020; Zhang et al. 2022) and imitation learning on existing heuristic branching rules (He et al. 2014; Yilmaz and Yorke-Smith 2021) have been developed for branching decisions in B&B algorithms. For work involving the use of MCTS to facilitate the search process in B&B, Sabharwal et al. (2012) propose a variant of Upper Confidence Bound for Trees (UCT) (Kocsis and Szepesvári 2006) to guide the search in a CPLEX solver at early stages, where rollouts are replaced by lower bounds obtained through LP relaxation. Recent work has focused on using the structure of the problem to combine MCTS with the B&B algorithm. For example, Baltussen et al. (2023) propose a parallel hybrid optimization algorithm that utilizes MCTS to guide the search in B&B for the vehicle routing problem, while Chour et al. (2023) use MCTS to search for feasible solutions, where an MILP, serving as a relaxation of the problem, provides lower bound information to reduce the search space. For work using MCTS in optimization problems, Loth et al. (2013) and Abe et al. (2019) extend MCTS to control the exploration of the search tree in constraint programming problems and graph-based optimization problems, respectively. Khalil et al. (2022) employ MCTS to solve the backdoor search problem in MILP, where a backdoor is a small subset of integer variables that allows the problem to be solved efficiently by branching only on those variables. Our work is distinct from the aforementioned studies. Instead of using MCTS to improve B&B search, our work focuses on using MCTS instead of B&B to solve MILPs, with the generation of the search space based on LP relaxation, which is similar to the B&B scheme. In addition, our work focuses on general MILPs, rather than focusing only on specific types of optimization problems. The most relevant work to our study is Fortin (2021), which is a preliminary study exploring the use of UCT to solve MILPs by combining heuristics implemented in the SCIP solver (Achterberg 2009). Unlike Fortin (2021), we consider a different MCTS algorithm and formally tailor it to solve MILPs without relying on modern solvers.

Our aim is to design an MCTS algorithm to efficiently solve MILPs. When using MCTS to solve MILPs, the search space comprises all branching candidate variables obtained from solving LPs. We formulate the decision-making process for tree search within this space as an MDP and adapt AOAT to solve MILPs. AOAT compares different nodes based on their corresponding posterior mean and posterior variance for the unknown state value. A node with a large posterior mean and/or a large posterior variance tends to be selected following the AOAT scheme, balancing exploration and exploitation. The advantages of AOAT over UCT are that AOAT tends to explore more, achieves asymptotic optimality, and has shown better performance (Zhang et al. 2022; Liu et al. 2023). A key challenge in using MCTS for MILPs is defining the appropriate reward for a leaf node, since it is unclear whether an obtained feasible solution is optimal. We propose a reward assignment mechanism that compares multiple leaf node values in a round before assigning rewards. Existing diving heuristics for B&B algorithms provide prior information for a newly added node during the expansion phase of MCTS, and can also serve as a rollout policy for MCTS. A simple numerical experiment demonstrates the potential benefits of the proposed method, indicating its potential for further research.

The paper is organized as follows. Section 2 introduces some basic concepts and formulates the problem. Section 3 provides an overview of MCTS and the proposed algorithm. Section 4 presents the experimental setup for the empirical validation of the proposed algorithm and discusses the empirical results. The paper concludes with some perspectives for further research.

2 PROBLEM FORMULATION

In this section, we describe the fundamental concepts relevant to this work and provide a formulation for searching within the tree-based space as an MDP.

2.1 Mixed Integer Linear Program (MILP)

An MILP is a mathematical optimization problem that consists of a set of linear constraints, a linear objective function, and variables that can be continuous or integral. Given a constraint coefficient matrix $A \in \mathbb{R}^{m\times n}$, a constraint right-hand-side vector $b \in \mathbb{R}^m$, an objective coefficient vector $c \in \mathbb{R}^n$, and a subset $I \subseteq \{1,\dots,n\}$, an MILP $= (A,b,c,I)$ can be defined as

$$z^* = \min\{c^T x | Ax \le b, x \in \mathbb{R}^n, x_j \in \mathbb{Z}, \forall j \in I\} \ . \tag{1}$$

Denote $X_{\text{MILP}} = \left\{x \in \mathbb{R}^n | Ax \le b, x \in \mathbb{R}^n, x_j \in \mathbb{Z}, \forall j \in I\right\}$ as the set containing feasible solutions of (1). A feasible solution $x^* \in X_{\text{MILP}}$ is optimal if its objective value satisfies $c^T x^* = z^*$. The linear programming (LP) relaxation of (1), derived by ignoring the integer constraints, is expressed as

$$\widetilde{z} = \min\{c^T x | x \in P_{\text{LP}}\} \ , \tag{2}$$

where $P_{\text{LP}} = \{x \in \mathbb{R}^n | Ax \le b, x \in \mathbb{R}^n\}$. Let $\bar{x}_R$ be the optimal solution for (2). Since $X_{\text{MILP}} \subseteq P_{\text{LP}}$, it follows that $\widetilde{z} \le z^*$, indicating that the optimal solution for (2) provides a lower bound for (1), also known as the dual bound. The LP is convex and can be solved efficiently using various algorithms, such as the simplex algorithm. If a solution to the LP relaxation satisfies the original integrality, then it also serves as a solution to (1).

2.2 Tree Structure of the Branch and Bound (B&B) Algorithm

Starting from a root node representing the LP relaxation of the original MILP, a B&B algorithm recursively constructs a search tree by branching fractional variables that need to be integers to reduce the feasibility region. Each node in the search tree represents an LP relaxation subproblem, and branching a variable at a node creates two child nodes. For a given LP relaxation subproblem Q, let $\bar{x}_Q$ represent the optimal solution for $\tilde{z}_Q = \min\{c^{\mathrm{T}}x | x \in Q\}$. For $\bar{x}_Q \notin X_{\mathrm{MILP}}$, let $A = \{i \in I \mid \bar{x}_Q^i \notin \mathbb{Z}\}$ be the set of branching candidate variables that do not meet the integrality constraint. A branching rule determines which variable $i^* \in A$ to branch on and in which direction (floor or ceiling) to proceed. This decision plays a vital role in the success of B&B: Q is decomposed into two subproblems, with additional constraints $x_Q^{i^*} \leq \lfloor \bar{x}_Q^{i^*} \rfloor$ and $x_Q^{i^*} \geq \lceil \bar{x}_Q^{i^*} \rceil$, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ represent the floor and ceiling functions, respectively. The two subproblems differ from the parent LP only in the variable bounds for $x_Q^{i^*}$.

The search tree ends when a feasible or infeasible solution is found, with the leaf nodes being referred to as fathomed nodes. A feasible solution at a fathomed node occurs when $\bar{x}_Q \in X_{\mathrm{MILP}}$, whereas an infeasible solution occurs when the LP is infeasible. For a minimization problem (1), the optimal solution of the LP provides a lower bound (i.e., dual bound) for the original MILP, whereas the current optimal feasible solution provides an upper bound (i.e., primal bound) for the original MILP. The solving process ends when the primal and dual bounds are equal or when the feasible regions can no longer be further decomposed, demonstrating optimality or infeasibility for the original MILP, respectively. Figure 1 illustrates a tree structure using B&B to solve an MILP.

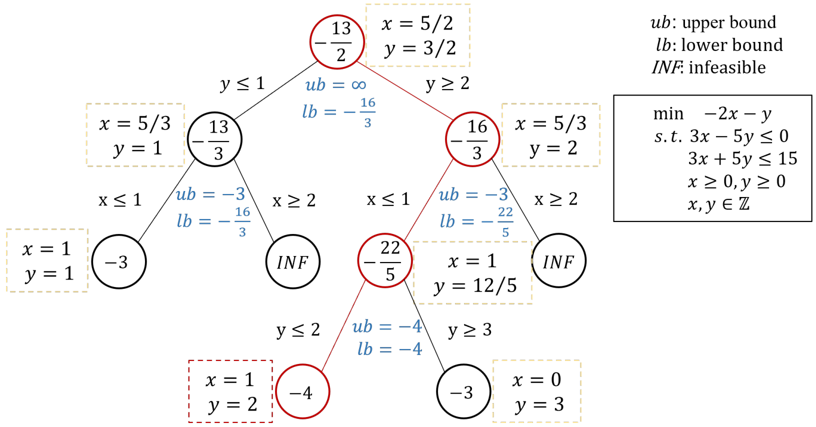

Figure 1: The search tree generated by using B&B to solve an MILP.

A diving heuristic starts from a given node and expands a single path in the depth-first order down the B&B tree until it reaches a leaf node, where branching decisions are made based on a certain criterion. Some commonly used diving heuristics are as follows:

- Random Diving (RD): selecting the branching variable at random and fixing it to a random direction.
- Fractional Diving (FD): selecting the variable with the lowest fractionality and fixing it in the corresponding direction, i.e., $i^* = \arg\min_{i \in A} |\bar{x}_Q^i - \lfloor \bar{x}_Q^i + 0.5 \rfloor|$. The rationale for FD is that the variable in the set A with a fractional value closest to an integer is more likely to converge to this integer value in the optimal solution.
- Line Search Diving (LSD): considering the ray originating from $\bar{x}_R$ and passing through $\bar{x}_Q$, this heuristic selects the variable whose coordinate hyperplane $x_Q^i = \lfloor \bar{x}_Q^i \rfloor$ or $x_Q^i = \lceil \bar{x}_Q^i \rceil$ is intersected first by the ray. The selected variable is given by: $i^* = \arg\min_{i \in A} s_i^{LSD}$, where $s_i^{LSD} = \frac{\bar{x}_Q^i - \lfloor \bar{x}_Q^i \rfloor}{\bar{x}_R^i - \bar{x}_Q^i}$ if $\bar{x}_R^i > \bar{x}_Q^i$, and $s_i^{LSD} = \frac{\lceil \bar{x}_Q^i \rceil - \bar{x}_Q^i}{\bar{x}_Q^i - \bar{x}_R^i}$, otherwise.
- Coefficient Diving (CD): selecting the variable with the minimal number of (positive) up-locks or down-locks and fixing it in the corresponding direction. A variable lock refers to the number of constraints that are violated by fixing a fractional variable to either its ceiling or floor value.

The B&B algorithm considers pruning suboptimal branches to keep the search tree and the computational steps small, thereby reducing the solving time and memory requirements. Typically, in the B&B algorithm, for a minimization problem, a node is pruned if the objective value $\widetilde{z}_Q$ of the corresponding LP relaxation problem is larger than or equal to the objective value of the currently obtained feasible solution.

2.3 Markov Decision Process (MDP) Formulation

The decision-making process for searching within the tree-based space composed of branching candidate variables can be formulated as an MDP, with MCTS serving as an efficient tool for solving the MDP by making a sequence of decisions. An MDP is typically expressed as a four-tuple $(\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R})$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, respectively. $\mathcal{P}$ is the Markovian transition model, with $p(s,a,s')$ representing the probability of transitioning to state s' after selecting action a in state s. $\mathcal{R}:\mathcal{S}\mapsto\mathbb{R}$ is the reward function. A policy $\pi:\mathcal{S}\mapsto\mathcal{A}$, which starts from an initial state and continues until reaching a terminal state or a defined time horizon, gathers a sum of rewards.

When solving MILPs using MCTS, each node in the search tree represents an LP relaxation problem, comprising (2) along with all constraints added from all past branching decisions. Each node generates a set of branching candidate variables, with each having two possible branching directions. The agent at a node determines a branching variable from these candidates and chooses a corresponding branching direction, resulting in a transition to a child node. An RL algorithm aims to learn an optimal policy π^*, which is a sequence of actions starting from the root node. To be specific, we define the following: **State** $\mathcal{S}$: A state $s_t\in\mathcal{S}$ represents the current LP, i.e., $s_t\overset{\Delta}{=}Q_t$, $t=0,\cdots,H$, with H indicating the horizon length; **Action** $\mathcal{A}$: The action set for a given state s_t is state-dependent, i.e., $\mathcal{A}_{s_t}\subseteq\mathcal{A}$. An action $a_t\in\mathcal{A}_{s_t}$ is a branching decision involving a branching candidate variable and a direction to fix, i.e., $\mathcal{A}_{s_t}\overset{\Delta}{=}\{A_t;\{\lfloor\cdot\rfloor,\lceil\cdot\rceil\}\}$; **Transition** $\mathcal{P}$: The transition is deterministic, i.e, $p(s_t,a_t,s_{t+1})=1$, $s_t,s_{t+1}\in\mathcal{S}$, $a_t\in\mathcal{A}_{s_t}$; and **Reward** $\mathcal{R}$: No reward is assigned to intermediate states, and the reward for the terminal state $R(s_H)$ is set based on whether the solution to Q_H is feasible or infeasible for the original MILP, indicating a sparse reward.

This MDP formulation is similar to that in the game of Go, where MCTS is used to efficiently explore large state spaces and make informed decisions based on simulations. Notice that this formulation differs from those in Gasse et al. (2019) and Mazyavkina et al. (2021), which model the decision-making process in the B&B tree as an MDP, where each state is a current B&B tree, providing an opportunity to use RL algorithms to learn branching decisions in B&B.

3 TAILORING MCTS TO MILP

This section introduces AOAT and presents several specific modifications made to tailor the algorithm for solving MILPs. The first issue concerns the assignment of the reward value at the leaf node in each MCTS rollout, which should effectively distinguish between feasible, infeasible, and optimal solutions. A second issue relates to the rollout policy in MCTS. Although MCTS can converge to the correct state-action value function given enough search budget, even if the heuristic used for the rollout policy is suboptimal, having a good rollout policy could make a significant difference in the performance of MCTS for MILPs.

3.1 Asymptotically Optimal Allocation for Trees (AOAT)

The AOAT algorithm simultaneously explores and builds a search tree, initially starting at its root node s_0, while updating an estimate of the state-action value function $\mathcal{Q}(s_0,a)$ across N tree-walks. Each tree-walk involves three phases: selection, expansion, and rollout.

The **selection** phase starts from the root node (initial state s_0) and iteratively selects a child node (state-action pair) until reaching a leaf node in the current MCTS tree. Node selection is modeled as an R&S problem. For a problem aiming to find an action for the root node with the maximal $\mathcal{Q}(s_0,a)$, the selected action a^* is determined by $a^* = \arg\max_{a\in\mathcal{A}_s} \mathcal{V}_{s,a}$, where for $\widetilde{a} = \arg\max_{a\in\mathcal{A}_s} \widehat{\mathcal{Q}}_{s,a}$, $\mathcal{V}_{s,\widetilde{a}} = \min\limits_{a\in\mathcal{A}_s,\, a\neq\widetilde{a}} \frac{\left(\widehat{\mathcal{Q}}_{s,\widetilde{a}}-\widehat{\mathcal{Q}}_{s,a}\right)^2}{\widetilde{\sigma}_{s,\widetilde{a}}+\widehat{\sigma}_{s,a}}$, and for $\widehat{a} \in \mathcal{A}_s$, $\widehat{a} \neq \widetilde{a}$, $\mathcal{V}_{s,\widehat{a}} = \min\left\{ \frac{\left(\widehat{\mathcal{Q}}_{s,\widetilde{a}}-\widehat{\mathcal{Q}}_{s,\widehat{a}}\right)^2}{\widehat{\sigma}_{s,\widetilde{a}}+\widetilde{\sigma}_{s,\widehat{a}}}, \min\limits_{a\in\mathcal{A}_s,\, a\neq\widetilde{a},\widehat{a}} \frac{\left(\widehat{\mathcal{Q}}_{s,\widetilde{a}}-\widehat{\mathcal{Q}}_{s,a}\right)^2}{\widehat{\sigma}_{s,\widetilde{a}}+\widehat{\sigma}_{s,a}} \right\}$. In addition, $\widehat{\sigma}_{s,a} = \left(\frac{1}{\widehat{\sigma}^0_{s,a}} + \frac{n_{s,a}}{\bar{\sigma}_{s,a}}\right)^{-1}$ represents posterior variance, and $\widehat{\mathcal{Q}}_{s,a} = \widehat{\sigma}_{s,a}\left(\frac{\widehat{\mathcal{Q}}^0_{s,a}}{\widehat{\sigma}^0_{s,a}} + \frac{n_{s,a}\bar{\mathcal{Q}}_{s,a}}{\bar{\sigma}_{s,a}}\right)$ represents posterior mean, where $\widehat{\mathcal{Q}}^0_{s,a}$ and $\widehat{\sigma}^0_{s,a}$ contain prior information. $n_{s,a}$ denotes the number of times action a has been selected in node s, and $\bar{\mathcal{Q}}_{s,a}$ and $\bar{\sigma}_{s,a}$ stand for the sample mean and variance collected when selecting action a from node s, respectively, $\widetilde{\sigma}_{s,a} = \left(\frac{1}{\widehat{\sigma}^0_{s,a}} + \frac{n_{s,a}+1}{\bar{\sigma}_{s,a}}\right)^{-1}$. The trade-off between exploration and exploitation is controlled by the posterior mean and variance for a certain state value. The difference in posterior means of two nodes $\widetilde{a}$, and $a \in \mathcal{A}_s$, $a \neq \widetilde{a}$ being small and their variances being large tends to lead to a small $\mathcal{V}_{s,a}$, and AOAT allows one of the two nodes that are most difficult to compare to be selected.

The **expansion** phase occurs once a state not in the tree is reached. This phase initializes $n_{s,a}$ with $n^0_{s,a}$, along with $\widehat{\mathcal{Q}}^0_{s,a}$ and $\widehat{\sigma}^0_{s,a}$. It then adds the current state to the tree and moves on to the rollout stage. The values for $\widehat{\mathcal{Q}}^0_{s,a}$ and $\widehat{\sigma}^0_{s,a}$ can be set based on prior expert knowledge of the problem. If no prior knowledge is available, these values can be initialized to a small default.

The **rollout** phase starts after the expansion stage, where actions are iteratively selected according to a certain rollout policy until reaching a terminal state s_H. The rollout policy is typically stochastic or heuristic, guiding the search toward promising areas. The resulting reward $R(s_H)$ is used to update $\widehat{\mathcal{Q}}_{s,a}$ and $\widehat{\sigma}_{s,a}$ in all nodes visited during the tree-walk:

$$\begin{aligned}
&n_{s,a} \leftarrow n_{s,a}+1; \quad \bar{\mu}_{s,a} = \bar{\mathcal{Q}}_{s,a};\\
&\bar{\mathcal{Q}}_{s,a} \leftarrow \bar{\mathcal{Q}}_{s,a} + (R(s_H) - \bar{\mathcal{Q}}_{s,a})/n_{s,a};\\
&\bar{\sigma}_{s,a} \leftarrow \frac{n_{s,a}-1}{n_{s,a}}\bar{\sigma}_{s,a} + \frac{1}{n_{s,a}}\left(R(s_H) - \bar{\mathcal{Q}}_{s,a}\right)\left(R(s_H) - \bar{\mu}_{s,a}\right)\ .
\end{aligned}$$

Simulations are run until a stopping criterion is met, often a fixed number of iterations (i.e., N). The action with the maximal $\mathcal{Q}(s_0,a)$ is executed. Compared to the commonly used UCT, the setup of the R&S problem aligns with that of MCTS, and the assumption of a normal distribution for the reward value in AOAT makes it suitable for general decision-making problems. AOAT incorporates more sample information, such as sample variances. A search tree generated using MCTS to solve the same minimization problem in Figure 1 is shown in Figure 2. As shown in Figure 2, two branching sequences lead to the optimal solution. A B&B tree can be considered as a special case of the MCTS search tree after pruning with certain branching rules. Although the search tree for MCTS is larger than that for B&B, MCTS serves as an efficient tool for exploring spaces with large state and action sets without relying on pre-trained models. This suggests that the advantage of solving MILPs with MCTS becomes more pronounced as the complexity of the problem increases or when efficient branching rules for the B&B algorithm are difficult to find.

3.2 Assigned Reward for Leaf Nodes

The search tree of MCTS for solving MILPs always ends up at a feasible or infeasible solution. For a current feasible solution, it is uncertain whether there exists another feasible solution with a better objective function value. The objective of MCTS is to maximize the probability of correctly selecting the optimal child node of the root node, and the reward assignment for the leaf node determines how the expected value of each child node is calculated, which affects the balance between exploration and exploitation in MCTS. Properly designed rewards guide MCTS toward optimal paths and help avoid getting stuck in low-reward regions, which is crucial for MCTS to solve MILPs. An appropriate reward assignment for leaf nodes

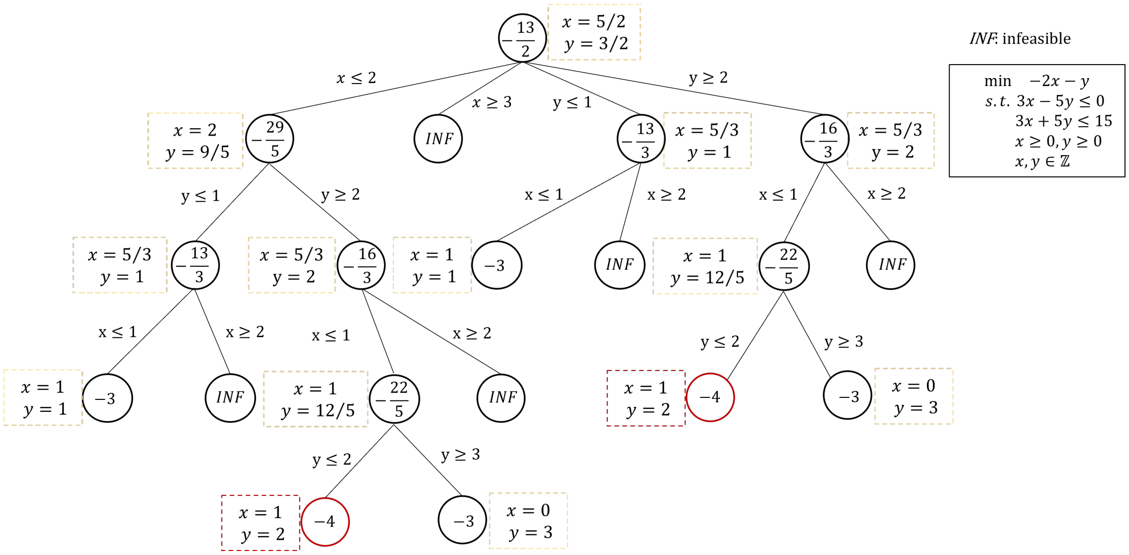

Figure 2: The search tree generated by using MCTS to solve an MILP, where the optimization considered is same in Figure 1.

should motivate MCTS to explore better feasible solutions, avoiding getting stuck in non-optimal feasible solutions and infeasible solutions.

A simple reward assignment is to set the reward as the corresponding objective function value for a feasible solution, and set a large enough penalty for an infeasible solution in a minimization problem (or a small enough penalty in a maximization problem), given that the optimal objective function value is unknown. However, if a child node of the root node that contains the optimal solution also includes many infeasible solutions at the leaf node, the estimation of its state value will converge to a degraded value, hindering the effectiveness of the MCTS. Therefore, simply using the objective function value may not result in a suitable reward assignment design for leaf nodes. Inspired by the reward assignment mechanism in using MCTS for computer games, obtaining a feasible solution at the leaf node can be considered a win in a game, whereas obtaining an infeasible solution can be considered a loss. Given that some MILP instances, such as the set cover problem (Chvatal 1979), always result in a feasible solution and different feasible solutions can yield different objective function values, with the goal being to find the best feasible solution, we propose a reward assignment method that evaluates whether a feasible solution is optimal among all feasible solutions in a round containing several MCTS simulations. This allows for assigning a loss to non-optimal feasible solutions, motivating MCTS to keep exploring the search space, avoiding settling for suboptimal solutions, and leading to the discovery of the truly optimal solution.

To be specific, $\ell \in \mathbb{Z}^+$ simulations of MCTS–comprising selection, expansion, and rollout–are run before backpropogating a reward $R(s_H)$. The hyperparameter ℓ can be tuned. Each set of ℓ simulations of MCTS is referred to as a round. For the first $(\ell-1)$ tree-walks in a round, only $n_{s,a} \leftarrow n_{s,a}+1$ is updated for all nodes visited during the tree-walk. For the ℓ-th tree-walk in a round, along with updating $n_{s,a}$, a reward of $R(s_H)=1$ is backpropagated from the leaf node s_H, corresponding to the optimal feasible solution among all ℓ solutions, up to the root node s_0. In addition, if the optimal feasible solution obtained in the current round is better than or equal to the optimal feasible solutions obtained in previous rounds, the reward is set to $R(s_H)=2$. This reward assignment for leaf nodes ensures that the child node with the highest state value at the root node is more likely to contain the better feasible solution, enabling MCTS to operate efficiently. Assigning only $n_{s,a}$ for the first $(\ell-1)$ tree-walks can be viewed as giving $R(s_H)=0$ for infeasible solutions and non-optimal feasible solutions, avoiding all ℓ simulations of MCTS being on the same tree-walk. As N increases, better feasible solutions tend to be included multiple times in a round of MCTS.

3.3 Heuristic initialization and rollouts

In the expansion phase of MCTS, when a new node is added to the search tree, certain node statistics and prior parameters are initialized. Prior information on the state of the problem can speed up the convergence of the state value. AOAT assigns a small $n_0 > 0$ to the initialized nodes, allowing reward to be obtained from leaf nodes, so that $\bar{\sigma}_{s,a}$ can be calculated, and empirically assigns $\widehat{Q}^0_{s,a}$ and $\widehat{\sigma}^0_{s,a}$. However, in MILPs, diving heuristics for B&B search determines which fractional variable to branch on and which direction to fix, providing a way to determine prior parameter $\widehat{Q}^0_{s,a}$ for each node. For example, to initialize $\widehat{Q}^0_{s,a}$ using information obtained from fractional diving, the value can be set to $\widehat{Q}^0_{s,a} = \frac{1}{\bar{x}^a_s - \lfloor \bar{x}^a_s \rfloor}$ or $\widehat{Q}^0_{s,a} = \frac{1}{\lceil \bar{x}^a_s \rceil - \bar{x}^a_s}$, depending on the corresponding fixing direction. To initialize $\widehat{Q}^0_{s,a}$ using information obtained from coefficient diving, the value can be set to the corresponding number of constraints that are violated.

As mentioned earlier, beyond the classical random rollout in MCTS, different diving heuristics can be used as an alternative for the rollout phase when solving MILPs. In addition, the bounding rule in the B&B algorithm offers a method to stop early in a tree-walk, thereby speeding up the MCTS search. Specifically, a tree-walk can be stopped when the LP relaxation solution is worse than the optimal feasible solution obtained so far, as this indicates that further progress cannot yield a better solution, resulting in a reward of zero for this node. The following pseudocode in Algorithm 1 illustrates the condensed form of the tailored AOAT for solving MILPs.

Algorithm 1 Tailored AOAT for Solving MILPs

Input: root node s_0 corresponding to LP relaxation of the original MILP; number of tree-walks N; algorithmic constants for AOAT: n_0, $\widehat{Q}^0_{s,a}$, $\widehat{\sigma}^0_{s,a}$; ℓ for number of simulations in a round.
Output: optimal feasible solution.

```
function AOAT-MILP($s_0$, $N$, $n_0$, $\widehat{Q}^0_{s,a}$, $\widehat{\sigma}^0_{s,a}$, $\ell$)
    $n \leftarrow 1$; $l \leftarrow 1$; feasol ← []
    while $n < N$ do
        $s_q \leftarrow$ SELECTIONPOLICY($s_0$, $n_0$, $\widehat{Q}^0_{s,a}$, $\widehat{\sigma}^0_{s,a}$)
        $s_H \leftarrow$ ROLLOUTPOLICY($s_q$)
        feasol ← the solution at $s_H$ (if it is a feasible solution)
        $n_{s,a} \leftarrow n_{s,a} + 1$;
        if $l == \ell$ then
            $s^*_H \leftarrow$ the best node corresponding to the optimal value in the set feasol;
            optfeasol ← the optimal value in feasol
            BACKPROPAGATE($s^*_H$, $R(s^*_H)$)
            $l \leftarrow 1$; feasol ← []
        else
            $l \leftarrow l + 1$
        end if
        $n \leftarrow n + 1$
    end while
    return optfeasol
end function

function SELECTIONPOLICY($s_0$, $n_0$, $\widehat{Q}^0_{s,a}$, $\widehat{\sigma}^0_{s,a}$)
    while $s$ is nonterminal do
        if $s_{q-1}$ is expandable then
            return EXPAND($s_{q-1}$)
        else
            $s_{t+1} \leftarrow$ AOAT($s_t$, $n_0$, $\widehat{Q}^0_{s,a}$, $\widehat{\sigma}^0_{s,a}$)
        end if
    end while
    return $\mathbf{s}_q$
end function

function EXPAND($s_{q-1}$)
  choose $a \in$ untried actions from $\mathcal{A}_{s_{q-1}}$
  append a new child $s_q$ to $s_{q-1}$
    return $s_q$
end function

function ROLLOUTPOLICY($s_q$)
    while $s$ is nonterminal do
        choose an action $a \in \mathcal{A}_s$ based on the rollout policy
        append a new child node $s' = (s,a)$ to the parent node $s$
        if the solution at $s'$ is worse than the solution corresponding to optfeasol then
            break
        end if
    end while
    return $s_H$
end function

function BACKPROPAGATE($s^*_H$,$R(s^*_H)$)
        update the node values in the corresponding tree-walk
end function
```

The proposed algorithm that uses AOAT to solve MILPs, referred to as AOAT-MILP, can be shown to be consistent, as demonstrated in Proposition 1, with the proof to be presented in future work.

Proposition 1. *The proposed AOAT-MILP is consistent, i.e., assuming the optimal solution to (1) exists, as $N \to \infty$, the optimal solution can definitely be found with the use of AOAT-MILP.*

4 NUMERICAL EXPERIMENTS

In this section, we demonstrate the empirical performance of the proposed AOAT-MILP algorithm on an optimization problem. We compare the proposed algorithm with UCT, algorithms using different rollout policies, and different initialization methods. The following are descriptions of the tested algorithms and their notations: AOAT-MILP (random+FDi): uses a random rollout policy and FD statistics to initialize $\widehat{Q}^0_{s,a}$; UCT-MILP (random+FDi): uses UCT as the MCTS algorithm, with the same initialization and rollout strategy as AOAT-MILP (random+FDi); AOAT-MILP (FD+FDi): uses FD as a rollout policy and FD statistics to initialize $\widehat{Q}^0_{s,a}$; AOAT-MILP (LSD+FDi): uses LSD as a rollout policy and FD statistics to initialize $\widehat{Q}^0_{s,a}$; AOAT-MILP (CD+FDi): uses CD as a rollout policy and FD statistics to initialize $\widehat{Q}^0_{s,a}$; AOAT-MILP (random+LSDi): uses a random rollout policy and LSD statistics to initialize $\widehat{Q}^0_{s,a}$.

For AOAT-MILP, we set $\widehat{\sigma}^0_{s,a} = 1$, $n_0 = 10$, $\ell = 1$, and if $\bar{\sigma}_{s,a} = 0$, a small $\varepsilon = 0.1$ is used as an alternative value. The exploration ratio in the UCT is set to $\sqrt{2}$. The tested optimization problem is as follows

$$\begin{aligned} \max \quad & 24x_1 + 13x_2 + 23x_3 + 15x_4 + 16x_5 \\ \text{s.t.} \quad & 12x_1 + 7x_2 + 11x_3 + 8x_4 + 9x_5 \le 26 \\ & x_1, \cdots, x_5 \in \{0,1\} \ , \end{aligned} \tag{3}$$

where its optimal solution is $x^* = (0,1,1,1,0)$ with an optimal objective function value of $z^* = 51$. The optimal solution to the LP relaxation problem of (3) is $\bar{x}_R = \left(1,0,1,\frac{3}{8},0\right)$ with an objective function value of $\widehat{z} = 52.625$. The optimization problem (3) is a special case of an MILP in which all integer variables are binary.

The search tree for solving (3) using MCTS can be enumerated, providing a more intuitive understanding of the numerical results. The number of tree-walks for each tested algorithm is set to $N = 80$, and the total number of tree-walks is 100 since there are two child nodes for the root node. The number of the tree-walks is intentionally not set too large to prevent fully exploring the tree, which would reduce the impact of the rollout policy. All tested algorithms can obtain the optimal solution x^* for (3), and we use the number of times x^* is visited as a performance measure. The results are obtained from 10^5 independent macro runs for each algorithm. Table 1 presents the visit frequency for x^*, along with the posterior state value for each of the two child nodes of the root node.

Table 1: The visit frequency for x^* and the $\widehat{Q}_{s,a}$ for each of the two child nodes of the root node (averaged over 10^5 runs).

	visit frequency for x^* (%)	$\widehat{Q}_{s,a}$ for $\bar{x}_R^4 = 0$	$\widehat{Q}_{s,a}$ for $\bar{x}_R^4 = 1$
AOAT-MILP (random+FDi)	17.43	0.6614	0.6078
UCT-MILP (random+FDi)	12.72	0.6432	0.5730
AOAT-MILP (FD+FDi)	21.87	0.8745	0.5667
AOAT-MILP (LSD+FDi)	30.88	0.9244	0.9391
AOAT-MILP (CD+FDi)	38.47	0.8018	1.1439
AOAT-MILP (random+LSDi)	17.46	0.6530	0.6065

From Table 1, we can observe that AOAT-MILP (random+FDi) achieves a higher visit frequency for the optimal solution than UCT-MILP (random+FDi) in the tested problem, demonstrating the efficiency of using AOAT instead of UCT as an MCTS algorithm for solving MILPs. We can also see that AOAT-MILP with various diving heuristics performs significantly better than AOAT-MILP with random diving, indicating that the rollout policy itself can influence the effectiveness of the AOAT-MILP algorithm. However, it is challenging to determine which rollout policy is best for general MILPs, as it may depend on the specific MILP instances being addressed. A random rollout policy could also perform well in complex, large-scale MILPs. Comparing AOAT-MILP (random+LSDi) with AOAT-MILP (random+FDi), we find that using different statistics from diving heuristics for initializing $\widehat{Q}^0_{s,a}$ also has an impact on the performance of the AOAT-MILP algorithm, although the effect is not as significant in the tested example, possibly because it influences the convergence of the AOAT-MILP.

Through enumeration, we find that the optimal solution occurs under the branch corresponding to $\bar{x}_R^4 = 1$ for the root node. From Table 1, we can observe that a higher visit frequency corresponds to a larger $\widehat{Q}_{s,a}$ for $\bar{x}_R^4 = 1$ compared to $\bar{x}_R^4 = 0$. This could be attributed to MCTS converging to the optimal state value for each node, indicating that the best action at the root node indeed contains the optimal solution x^* under the proposed reward assignment mechanism. We also set $\ell = 2$ for AOAT-MILP (random+FDi) in an additional experiment to demonstrate the effectiveness of our reward assignment mechanism, resulting in a higher visit frequency for x^*, i.e., 18.52. Both findings demonstrate the effectiveness of our reward assignment mechanism.

In our experiment, we do not use solving time as a performance measure for the different tested algorithms, as it is hardware-dependent and reliant on the testing environment. Using other performance measures, such as the primal integral and the gap between the primal and dual bounds, to gauge the efficiency of the tested algorithms, as well as testing on more challenging optimization problems and comparing with traditional B&B algorithms and modern solvers, are left for future work.

5 CONCLUSIONS

The paper explores the use of Monte Carlo Tree Search (MCTS) to solve Mixed Integer Linear Programs (MILPs). Following the Branch and Bound approach, we formulate solving MILP instances as a Markov decision process, allowing MCTS to efficiently address the problem. By setting appropriate rewards and using existing diving heuristics, we adapt the Asymptotically Optimal Allocation for Tree (AOAT) algorithm to solve MILPs. The proposed algorithm is tested on a simple problem to demonstrate its effectiveness.

This work demonstrates that MCTS has potential for solving MILPs, offering various avenues for further research. Evaluating the performance of the proposed method on complex MILP instances and comparing its efficiency with that of modern solvers deserve future work. Further research also include developing a more effective reward assignment mechanism and finding ways to avoid solving a linear programming (LP) relaxation problem after each branching decision at a node. In the search tree for solving MILPs with MCTS, pruning nodes would not eliminate the possibility of finding the optimal solution, as multiple sequences of decisions can lead to the optimal solution, and thus exploring strategies for pruning nodes to reduce the search space in complex problems also deserves future work. How to train a policy neural network to learn the probability distribution for each action to improve the overall performance of the proposed algorithm could also be future work. MCTS has proven efficient in solving complex problems with large state and action spaces, suggesting that it could be used to tackle challenging MILP instances where even modern solvers struggle to find good feasible solutions.

ACKNOWLEDGMENTS

The work was supported by the National Natural Science Foundation of China (NSFC) under Grants 72325007, 72250065, and 720220011, and a PKU-Boya Postdoctoral Fellowship 2406396158.

REFERENCES

Abe, K., Z. Xu, I. Sato, and M. Sugiyama. 2019. “Solving NP-Hard Problems on Graphs with Extended Alphago Zero”. *arXiv preprint arXiv:1905.11623*.

Achterberg, T. 2007. “Constraint Integer Programming”. *Ph. D. Thesis, Technische Universitat Berlin*.

Achterberg, T. 2009. “SCIP: Solving Constraint Integer Programs”. *Mathematical Programming Computation* 1:1–41.

Achterberg, T., T. Koch, and A. Martin. 2005. “Branching Rules Revisited”. *Operations Research Letters* 33(1):42–54.

Applegate, David and Bixby, Robert 1995. “Finding Cuts in the TSP (A Preliminary Report)”.

Baltussen, T., M. Goutham, M. Menon, S. G. Garrow, M. Santillo and S. Stockar. 2023. “A Parallel Monte-Carlo Tree Search-Based Metaheuristic for Optimal Fleet Composition Considering Vehicle Routing Using Branch & Bound”. In *2023 IEEE Intelligent Vehicles Symposium*, 1–6. Anchorage, Alaska, USA: Institute of Electrical and Electronics Engineers, Inc.

Bénichou, M., J.-M. Gauthier, P. Girodet, G. Hentges, G. Ribière and O. Vincent. 1971. “Experiments in Mixed-Integer Linear Programming”. *Mathematical Programming* 1:76–94.

Chour, K., P. Pradeep, A. A. Munishkin, and K. M. Kalyanam. 2023. “Aerial Vehicle Routing and Scheduling for UAS Traffic Management: A Hybrid Monte Carlo Tree Search Approach”. In *Proceedings of the 2023 IEEE/AIAA Digital Avionics Systems Conference*, 1–9. Barcelona, Spain: Institute of Electrical and Electronics Engineers, Inc.

Chvatal, V. 1979. “A Greedy Heuristic for the Set-Covering Problem”. *Mathematics of Operations Research* 4(3):233–235.

Dakin, R. J. 1965. “A Tree-Search Algorithm for Mixed Integer Programming Problems”. *The Computer Journal* 8(3):250–255.

Dantzig, G. B. 1951. “Maximization of a Linear Function of Variables Subject to Linear Inequalities”. *Activity Analysis of Production and Allocation* 13:339–347.

Enzenberger, M., M. Müller, B. Arneson, and R. Segal. 2010. “Fuego—An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search”. *IEEE Transactions on Computational Intelligence and AI in Games* 2(4):259–270.

Fortin, V. 2021. *Monte Carlo Tree Search as a Primal Heuristic for Integer Programming*. Ph. D. thesis, HEC Montréal.

Fu, M. C. 2018. “Monte Carlo Tree Search: A Tutorial”. In *2018 Winter Simulation Conference (WSC)*, 222–236 https://doi.org/10.1109/WSC.2018.8632344.

Gasse, M., D. Chételat, N. Ferroni, L. Charlin and A. Lodi. 2019. “Exact Combinatorial Optimization with Graph Convolutional Neural Networks”. *Advances in Neural Information Processing Systems* 32:1–13.

He, H., H. Daume III, and J. M. Eisner. 2014. “Learning to Search in Branch and Bound Algorithms”. *Advances in Neural Information Processing Systems* 27:1–9.

Huang, L., X. Chen, W. Huo, J. Wang, F. Zhang, B. Bai *et al*. 2021. “Branch and Bound in Mixed Integer Linear Programming Problems: A Survey of Techniques and Trends”. *arXiv preprint arXiv:2111.06257*.

Khalil, E. B., P. Vaezipoor, and B. Dilkina. 2022. “Finding Backdoors to Integer Programs: A Monte Carlo Tree Search Framework”. In *36th Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 36, 3786–3795. Vancouver, BC, Canada.

Kocsis, L. and C. Szepesvári. 2006. “Bandit Based Monte-Carlo Planning”. In *European Conference on Machine Learning*, 282–293. Berlin, Germany: Springer.

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

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

Liu, X., Y. Peng, G. Zhang, and R. Zhou. 2023. “An Efficient Node Selection Policy for Monte Carlo Tree Search with Neural Networks”. *Available at SSRN 4450999*.

Loth, M., M. Sebag, Y. Hamadi, and M. Schoenauer. 2013. “Bandit-Based Search for Constraint Programming”. In *19th International Conference on Principles and Practice of Constraint Programming*, 464–480. Uppsala, Sweden: Springer.

Mazyavkina, N., S. Sviridov, S. Ivanov, and E. Burnaev. 2021. “Reinforcement Learning for Combinatorial Optimization: A Survey”. *Computers & Operations Research* 134:105400.

Nair, V., S. Bartunov, F. Gimeno, I. Von Glehn, P. Lichocki, I. Lobov, , , , *et al*. 2020. “Solving Mixed Integer Programs Using Neural Networks”. *arXiv preprint arXiv:2012.13349*.

Paulus, M. and A. Krause. 2024. “Learning to Dive in Branch and Bound”. *Advances in Neural Information Processing Systems* 36:1–18.

Peng, Y., E. K. Chong, C.-H. Chen, and M. C. Fu. 2018. “Ranking and Selection as Stochastic Control”. *IEEE Transactions on Automatic Control* 63(8):2359–2373.

Sabharwal, A., H. Samulowitz, and C. Reddy. 2012. “Guiding Combinatorial Optimization with UCT”. In *9th International Conference on Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, 356–361. Nantes, France: Springer.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, , , , *et al*. 2017. “Mastering the Game of Go Without Human Knowledge”. *Nature* 550(7676):354–359.

Świechowski, M., K. Godlewski, B. Sawicki, and J. Mańdziuk. 2023. “Monte Carlo Tree Search: A Review of Recent Modifications and Applications”. *Artificial Intelligence Review* 56(3):2497–2562.

Tang, Y., S. Agrawal, and Y. Faenza. 2020. “Reinforcement Learning for Integer Programming: Learning to Cut”. In *37th International Conference on Machine Learning*, 9367–9376. Vienna, Austria: PMLR.

Wolsey, L. A. and G. L. Nemhauser. 2014. *Integer and Combinatorial Optimization*. John Wiley & Sons.

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

Zhang, G., Y. Peng, and Y. Xu. 2022. “An Efficient Dynamic Sampling Policy for Monte Carlo Tree Search”. In *2022 Winter Simulation Conference (WSC)*, 2760–2771 https://doi.org/10.1109/WSC57314.2022.10015374.

Zhang, G., Y. Peng, J. Zhang, and E. Zhou. 2023. “Asymptotically Optimal Sampling Policy for Selecting Top-m Alternatives”. *INFORMS Journal on Computing* 35(6):1261–1285.

Zhang, T., A. Banitalebi-Dehkordi, and Y. Zhang. 2022. “Deep Reinforcement Learning for Exact Combinatorial Optimization: Learning to Branch”. In *26th International Conference on Pattern Recognition*, 3105–3111. Montréal Québec, Canada: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

GONGBO ZHANG is an assistant research fellow at the Guanghua School of Management, Peking University, Beijing, China. His research interests include stochastic modeling and analysis, simulation optimization, and reinforcement learning. His email address is gongbozhang@pku.edu.cn.

YIJIE PENG is an Associate Professor in Guanghua School of Management at Peking University, Beijing, China. His research interests include stochastic modeling and analysis, simulation optimization, machine learning, data analytics, and healthcare. His email address is pengyijie@pku.edu.cn.