Reinforcement Learning for Variable Selection in Branch and Bound

Flynn Dowey [redacted]

Francis Chambers [redacted]

Abishek Bupathi [redacted]

Abstract

Optimization problems are prevalent in operations research and have many applications. Integer programs are an optimization problem designed to be combinatorial, i.e., non-convex, so traditional solvers cannot solve them. Branch and Bound (B&B) is an exact algorithm for solving an integer program called mixed integer linear programs (MILP)s. Variable selection, node selection, and pruning rules highly influence the performance of B&B. Configuring B&B to a time-realizable algorithm requires expert knowledge and is problem-specific. We propose using Reinforcement Learning (RL) to augment an agent that learns branching and pruning strategies in B&B with Graph Convolutional Neural Networks (GCNNs) trained with an off-policy policy gradient. Our agent learns solely from experience and is not taught by an expert. We aim to generalize the agent to be able to solve an MILP, regardless of its formulation.

1 Introduction

Combinatorial optimization problems are a cornerstone of operations research and machine learning, encompassing challenges such as the traveling salesman problem, scheduling and integer programming. These problems, characterized by their exponential solution spaces, often require advanced techniques to identify optimal solutions efficiently. Among these techniques, the branch-and-bound (B&B) algorithm stands out as the most commonly used method for solving combinatorial optimization problems. B&B iteratively partitions the search space by branching on decision variables, solving more manageable subproblem (by solving LP relaxations where the integer variables are allowed to take continuous values), and using upper and lower bounds to eliminate infeasible regions. More specifically, the algorithm uses the subproblem's LP solution as a lower bound and prunes or fathoms branches that cannot improve the current best-known solution given by the global feasible upper bound, effectively reducing the search space. B&B continues this process until all branches are explored or pruned, ultimately converging on the optimal solution [9]. However, its effectiveness heavily depends on the branching and pruning strategies employed, as poor decisions can lead to significant computational overhead.

Traditionally, branching and pruning in B&B rely on heuristic or manually designed rules. While these rules provide robust performance in general cases, they often fall short when applied to specific problem instances or distributions, leading to suboptimal decisions. To address this limitation, researchers have increasingly turned to machine learning techniques that can learn adaptive policies tailored to the structure of the problem. Recent advancements in supervised learning and reinforcement learning have demonstrated significant potential in optimizing these decisions, resulting in faster convergence and reduced computational overhead.

This paper explores a reinforcement learning approach that employs a Graph Convolutional Neural Network (GCNN)-based actor-critic architecture within an off-policy policy gradient framework. The actor learns a branching policy for selecting variables, while the critic estimates the state

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

value to guide the actor's learning by providing feedback on the policy's performance. The GCNN processes a bipartite graph representation of the optimization problem, where the graph consists of two types of nodes: variable nodes and constraint nodes. Using custom message-passing layers, the GCNN captures the relationships between variables and constraints, encoding them into meaningful embeddings. These embeddings enable the actor to make more informed and effective branching decisions, improving the overall efficiency of the B&B process.

2 Related Work

The first attempt to incorporate machine learning techniques to aid the branch and bound (B&B) paradigm was He u. a. [8], who learned a node selection and node pruning policy (not variable selection as in our scenario) using imitation learning. He u. a. [8] assume that all problem instances are solved *a priori* to which an oracle can guide the agent through branches of the B&B search tree where the optimal solution is guaranteed to be found. The downside is obviously that we require that a small set of solved problems are given at training time and the problems to be solved at test time are of the same type. Essentially we repeatedly retrain the policy to make decisions that agree better with the oracle's decisions so it will only perform well on test data similar to training data. In addition obtaining a large amount of high quality training data in this scenario is difficult and computationally expensive.

Other works which build upon the idea of imitation learning but use a strong branching oracle include Gasse u. a. [7]. Gasse u. a. [7] is one of the first papers to suggest modeling the state as a bipartitite graph (constraints and variables are the two types of nodes) with an edge connecting them if the variable is part of a given constraint and one of the first papers to parameterize the variable selection policy as a Graph Convolutional Neural Network (GCNN) which we also do in our paper. The GCNN learns how to encode the structure of the MILP (variables and constraints) into node representations and then the policy applies a final masked softmax activation on the variable nodes to produce a probability distribution over branching variables (ignoring variables that cannot be branched). The downside is as with all imitation learning models, a requirement for labeled strong branching training data which is not straightforward to obtain.

The alternative to imitation learning is reinforcement learning. More recent attempts to incorporate reinforcement learning for variable selection include Parsonson u. a. [15] and Tang u. a. [21]. In general, Reinforcement Learning (RL) algorithms for learning B&B cuts strugglge with sprase rewards (good branching decisions often result in rewards only after many steps) and difficult exploration (finding good branching strategies requires exploring a large, complex search space). Parsonson u. a. [15] propose Retro-Branching, an RL framework to improve branching decisions where the algorithm retrospectively decompose the search tree of B&B into multiple shorter paths (sub-trees). Shorter trajectories enable the RL agent to learn from more predictable next states, addressing the issues of sparse rewards and difficult exploration. Nevertheless, Retro-Branching achieves only 80% of the performance of the best IL methods on benchmark MILPs. Despite eliminating the need for expensive expert data, the method does not yet outperform imitation learning in terms of solution quality or efficiency.

Tang u. a. [21] demonstrates the potential of reinforcement learning for improving cutting-plane selection, a critical component of solving mixed-integer linear programs (MILPs). Their approach leverages deep RL to adaptively select cutting planes, significantly outperforming traditional human-designed heuristics and generalizing to larger instances and problem classes. Despite its success, the method focuses specifically on cut selection and does not address the broader challenges of variable selection or branching decisions within the B&B framework.

In our work, we propose an off-policy policy gradient (OFF-PG) approach that addresses these challenges by utilizing off-policy learning to improve branching decisions. Unlike on-policy RL methods, which suffer from sparse rewards and difficult exploration, OFF-PG leverages pre-collected trajectories generated by a behavioral policy to stabilize training and improve sample efficiency. By incorporating importance sampling, generalized advantage estimation (GAE), and actor-critic networks, our method enables effective learning of branching strategies without requiring expensive expert data or extensive trial-and-error. As a result, our approach achieves competitive or superior solution quality compared to imitation learning methods while also offering greater scalability and flexibility across diverse MILP instances.

3 Background

3.1 Mixed Integer Linear Programs

A Mixed Integer Linear Program (MILP) is an extension of a Linear Program (LP) by incorporating integer constraints only on some variables while others remain continuous. It retains the linear objective function and constraints, allowing to address complex real-world challenges in fields like logistics, scheduling, and resource allocation where both discrete and continuous optimization are essential. The canonical form of an MILP can be written as

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & l \leq x \leq u \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \end{aligned} \tag{1}$$

where $c \in \mathbb{R}^n$ is the objective coefficient, $A \in \mathbb{R}^{m \times n}$ is the constraint matrix, $b \in \mathbb{R}^m$ is the constraint vector, $u, l \in \mathbb{R}^n$ are upper and lower bounds on x, and $p \leq n$ is the number of integer variables. A feasible solution to (1) is one that satisfies all constraints and an optimal solution is feasible and is a minimizer of (1). A lower bound, z can be obtained by relaxing the integer constraints and solving the corresponding LP.

3.2 The Branch and Bound Algorithm

Methods to solve MILPs include exact algorithms, heuristic methods, and a special case of exploiting total unimodularity. Branch and Bound (B&B) [2] is an exact algorithm that systematically partitions the optimization problem into smaller sub-problems, using a bounding function to eliminate suboptimal branches and narrow the search space.

The root node in a B&B tree is the relaxed LP of (1) where all variables can be continuous. If a solution to the LP relaxation respects the original integrality constraints, then it is a solution to (1); otherwise, the search is extended. Candidate LPs are formed by the following binary decomposition,

$$x_i \leq \lfloor x_i^* \rfloor \cap x_i \geq \lceil x_i^* \rceil, \quad \forall i \in [p] : x_i^* \notin \mathbb{Z} \tag{2}$$

and the search continues. One can observe that two decisions must be made: (1) the choice of which variable to append constraints on, x_i - called the branching variable - and (2) which node to visit next in the search. The remainder of this paper focuses on the branching problem.

4 Method

We now discuss our approach to solving the B&B variable selection problem in MILPs. We formulate the problem as a Partially Observable Markov Decision Process (POMDP) and apply reinforcement learning (RL).

States The agent at time t will be given local observations of the B&B tree, denoted by s_t. Each node in the tree can be represented as a bipartite graph, as proposed by [7]. Concretely, the set of m constraints - corresponding to the rows of A in (1) - can be formulated as a feature matrix $C \in \mathbb{R}^{m \times d_c}$. Similarly, the set of n variables - corresponding to the columns of A in (1) - can be represented by $V \in \mathbb{R}^{n \times d_v}$. An edge $(i, j) \in \mathcal{E}$ if and only if $A_{i,j} \neq 0$. Edges are represented by edge features $E \in \mathbb{R}^{m \times n \times d_e}$. The agent reaches a terminal state if either of the following occurs: the agent solves the MILP, the problem is infeasible, or the allotted solving time expires.

Actions The set of actions available at time t, denoted $\mathcal{A}(s_t)$, are represented by the $k \leq p$ fractional variables to branch on, i.e. $a_t \in \mathcal{A}(s_t) \subseteq \{1, \ldots, k\}$, in the relaxed LP.

Rewards The reward, $r(s_t, a_t, s_{t+1})$ the agent will receive after observing s_t, branching on variable a_t, and subsequently observing the next state s_{t+1}, will be the negative change in area over the dual bound.

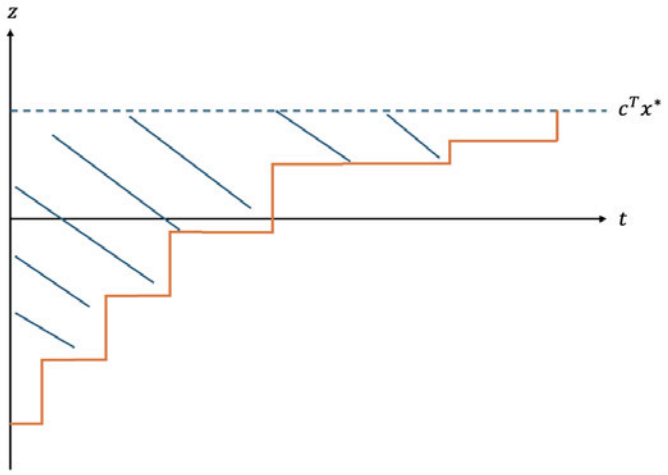

Figure 1: Illustration of the dual bound z (orange) converging to the optimal solution $c^T x^*$ over time, t. The dual integral is the blue area shaded above the dual bound.

$$r(s_t, a_t, s_{t+1}) = -\int_t^{t+1} z_\tau^* d\tau \tag{3}$$

The reward formulated in (3) is derived from the dual integral [16], where z_τ^* is the best dual bound found so far. The dual bound corresponds to a relaxed LP of the original MILP. By branching, the LP relaxations corresponding to the branch-and-bound tree leaves get tightened, and the dual bound increases over time [6] - refer to Figure 1. Using this reward motivates the agent to make good and fast decisions, with the hope that the agent converges to tight optimality guarantees.

Transition Probabilities The probability of a trajectory $\tau = (s_0, a_0, r_0, \ldots, s_T) \in \mathcal{T}$ depends on the branching strategy π and the decisions made by the solver

$$p(\tau|\pi) = p(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{4}$$

Objective The objective is to maximize the sum of discounted rewards by learning an optimal policy π^*, shown below

$$\pi^* \in \arg\max_\pi \mathbb{E}_{\tau\sim\pi}\Big[\sum_t \gamma^t r(s_t, a_t, s_{t+1})\Big] \tag{5}$$

A natural solution is to find an estimate of the optimal policy by learning a parametrization where any form of gradient ascent can be performed. In reinforcement learning literature, these methods are called policy-gradient methods.

4.1 Off-Policy Policy Gradient

Instead of using on-policy methods, we propose using an off-policy policy gradient method, denoted OFF-PG. Off-policy methods have a wider range of applications and learning possibilities, notably the ability to learn from demonstration [5, 20]. Off-policy is advantageous for complex problems where learning from scratch is intractable due to the extended horizon of an episode and the potential for sparse or uninformative reward signals [19].

Our method consists of a behavioural policy, $b_\psi(a|s)$, an actor $\pi_\theta(a|s)$ and a critic $V_\phi(s)$ each of which is parameterized by a GCNN. The behavioural policy attempts to mimic strong branching using the method proposed in [7]. The actor and critic are loaded with pre-trained weights from the

behavioural policy to accelerate learning. The actor and critic can learn all network parameters except message-passing layers to lower memory consumption.

We utilize the generalized advantage estimate (GAE) [17] when training and multiply the actor loss by a weighted importance sampling ratio $\pi_\theta(a|s)/b_\psi(a|s)$ outlined in [5]. Furthermore, we follow the same strategy outlined in the proximal policy optimization (PPO) algorithm [18], where we limit the magnitude of the weighted sampling factor. We introduce soft updates to the critic using a target critic network to minimize variance. The weights of the target network are updated according to a Polyak step size of $\tau = 0.01$, specifically

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$

where ϕ' are the target parameters and ϕ are the parameters of the actual critic. This approach is used in other policy gradient methods like Deep Deterministic Policy Gradient (DDPG) [11].

Refer to algorithm 1 for a outline of OFF-PG.

Algorithm 1 Overview of OFF-PG training algorithm

procedure OFF-PG
  Train the behavioral policy b_ψ using the method in [7]
  Load the pre-trained weights to the actor and critic and freeze convolutional layers
  for $k = 0, 1, 2, \ldots$ **do**
    Collect a trajectory $\mathcal{B} = \{\tau_i\}$ by running b_ψ
    Compute the rewards to go $\hat{R}_t$
    Compute advantages $\hat{A}_t$ using GAE and the current estimate of V_{ϕ_k}
    Update the policy by maximizing

$$\theta_{k+1} \in \arg\max_\theta \frac{1}{T}\sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{b_\psi(a_t|s_t)}\hat{A}_t, g(\epsilon, \hat{A}_t)\right)$$

    where $g(\epsilon, A) = \begin{cases}(1+\epsilon)A & A \geq 0\\ (1-\epsilon)A & A < 0\end{cases}$
    Update the value function V_ϕ by minimizing

$$\phi_{k+1} \in \arg\min_\phi \frac{1}{T}\sum_{t=0}^{T}\left(V_\phi(s_t) - \hat{R}_t\right)^2$$

  end for
end procedure

5 Experiments

5.1 Description

Benchmarks Our first benchmark consists of Set Covering instances [1] with 500 columns and a density of 0.05. We train on instances with 500 rows and evaluate all methods on 500 rows (easy), 600 rows (medium) and 700 rows (hard). Our second benchmark consists of instances related to the Capacitated Facility Location problem [4] with 50 facilities. We train with 100 customers and test on 100 customers (easy), 150 customers (medium) and 200 customers (hard).

Baselines We compare against version 8.1 of PySCIPOpt [12], which uses a variant of hybrid pseudo branching (HPB), and an imitation learning method proposed in [7] denoted as (GCNN). We neglect to include a full strong brancher (FSB) as it is not competitive in terms of running time, despite producing tiny search trees [7].

Training The behaviour policy is trained using the method outlined in [7], with 10,000 training samples and 2,000 validation samples generated randomly using different seeds across an arbitrary number of problem instances. This is to assert the reproducibility of the ablation study presented in

Section 5.3. The environment used to interact with the behaviour policy is randomly generated and distinct; this is to assert novel interactions with the agent.

The interactions obtained by the behaviour policy and the environment are stored in a replay buffer of 1,000 transitions; if the buffer exceeds capacity, the oldest entry is dropped. We do not discount the rewards, i.e. $\gamma = 1$, as this is an episodic task, and λ is set to 0.95 for GAE. The agent has a maximum time limit of 15 minutes to solve each problem instance. We allow the agent to see a maximum of 100 episodes and stop training if the actor or critic loss does not improve.

Evaluation Evaluation is conducted for each problem difficulty (Easy, Medium, Hard) on ten new instances using three different seeds, resulting in 30 solving attempts per method. We report standard metrics for MILP benchmarking [13], that is, the 1-shifted geometric mean of the solving times in seconds (Time) and the final node counts of instances (Nodes). We also report the number of (Wins) each baseline method achieves for each metric. Policy evaluation of Set Covering instances are presented in Table 1 and Capacitated Facility Location instances in 2. Note that we also report the standard error percentage, so "$a \pm b\%$ nodes" means it took on average a nodes to solve an instance, and when solving one of those instances, the number of nodes varied from the sample mean by $b\%$ on average. The best-performing metric is in bold.

Model	Easy			Medium			Hard		
	Time	Wins	Nodes	Time	Wins	Nodes	Time	Wins	Nodes
HPB	$3.78 \pm 28.16\%$	$6/30$	$381.10 \pm 41.05\%$	$4.32 \pm 18.60\%$	$7/30$	$339.97 \pm 23.14\%$	$\mathbf{6.43} \pm 7.41\%$	$\mathbf{17}/30$	$727.83 \pm 20.43\%$
GCNN	$\mathbf{3.61} \pm 33.38\%$	$\mathbf{12}/30$	$\mathbf{325.50} \pm 33.86\%$	$\mathbf{4.22} \pm 19.99\%$	$\mathbf{13}/30$	$\mathbf{299.43} \pm 20.02\%$	$8.09 \pm 5.06\%$	$7/30$	$609.13 \pm 18.28\%$
OFF-PG	$3.71 \pm 31.15\%$	$\mathbf{12}/30$	$344.80 \pm 35.18\%$	$4.39 \pm 17.97\%$	$10/30$	$309.13 \pm 19.33\%$	$7.78 \pm 5.42\%$	$6/30$	$\mathbf{601.47} \pm 18.92\%$

Table 1: Performance comparison on Set Covering problem.

Model	Easy			Medium			Hard		
	Time	Wins	Nodes	Time	Wins	Nodes	Time	Wins	Nodes
HPB	$10.91 \pm 3.06\%$	$7/30$	$\mathbf{162.87} \pm 20.47\%$	$20.55 \pm 1.19\%$	$3/30$	$\mathbf{151.33} \pm 17.29\%$	$26.82 \pm 0.82\%$	$1/30$	$\mathbf{86.07} \pm 16.53\%$
GCNN	$9.23 \pm 4.23\%$	$8/30$	$234.27 \pm 20.11\%$	$15.09 \pm 2.02\%$	$\mathbf{14}/30$	$248.33 \pm 19.43\%$	$15.69 \pm 1.75\%$	$\mathbf{17}/30$	$135.70 \pm 16.73\%$
OFF-PG	$\mathbf{8.35} \pm 4.99\%$	$\mathbf{15}/30$	$210.53 \pm 20.31\%$	$\mathbf{13.96} \pm 2.11\%$	$13/30$	$218.73 \pm 16.17\%$	$\mathbf{15.08} \pm 1.80\%$	$12/30$	$127.10 \pm 15.47\%$

Table 2: Performance comparison on Capacitated Facility Location.

5.2 Comparative experiment

At solving time, we see that OFF-PG's performance is subject to the problem instance, suggesting the need for hyper-parameter adjustment based on the problem type. For example, it is superior in expected solving time for Capacitated Facility Location across all seeds but underachieves in Set Covering. We observe that expected solving time differentiates from the number of wins a model can achieve. Interestingly, the best method for nodes, at least for the Capacitated Facility Location, is not necessarily the best in total solving time. Solving time includes the computational cost of the feature extraction and inference time.

OFF-PG shows the ability to generalize on Capacitated Facility Location problem instances but could benefit from training on a more diverse set of problem types or difficulty levels. We observe that on Set Covering problem instances, OFF-PG generalizes rather poorly, suggesting that the model's ability to generalize highly correlates with the type of MILP it is designed to solve.

5.3 Ablation study

We present an ablation study on our off-policy algorithm on the set covering problem by comparing three different reward functions: the negated difference in the dual integral since the previous state, the negated total number of nodes processed since the previous state, and the negative change in the primal integral since the previous state.

Table 3 presents results on Set Covering instances. A precise observation is the consequence of performance time and the choice of reward signal provided to the agent during learning. This observation is not reflected in the loss curves shown in 2, where the actor receiving the primal reward obtains the lowest loss, and the actor receiving the number of nodes reward fails to minimize the loss. This highlights the complexity of training an off-policy agent.

Reward	Easy			Medium			Hard		
	Time	Wins	Nodes	Time	Wins	Nodes	Time	Wins	Nodes
Dual Integral	$3.71 \pm 31.15\%$	$\mathbf{10}/30$	$344.80 \pm 35.18\%$	$4.39 \pm 17.97\%$	$5/30$	$309.13 \pm 19.33\%$	$7.78 \pm 5.42\%$	$5/30$	$601.47 \pm 18.92\%$
Primal Integral	$\mathbf{3.53} \pm 35.22\%$	$\mathbf{10}/30$	$\mathbf{319.47} \pm 32.75\%$	$\mathbf{4.13} \pm 21.07\%$	$8/30$	$\mathbf{292.67} \pm 18.60\%$	$7.88 \pm 5.31\%$	$7/30$	$608.93 \pm 18.37\%$
Number of Nodes	$3.55 \pm 34.61\%$	$\mathbf{10}/30$	$331.80 \pm 32.98\%$	$4.24 \pm 19.89\%$	$\mathbf{17}/30$	$338.13 \pm 20.39\%$	$\mathbf{7.14} \pm 6.24\%$	$\mathbf{18}/30$	$\mathbf{594.93} \pm 18.77\%$

Table 3: Performance comparison on different reward signals.

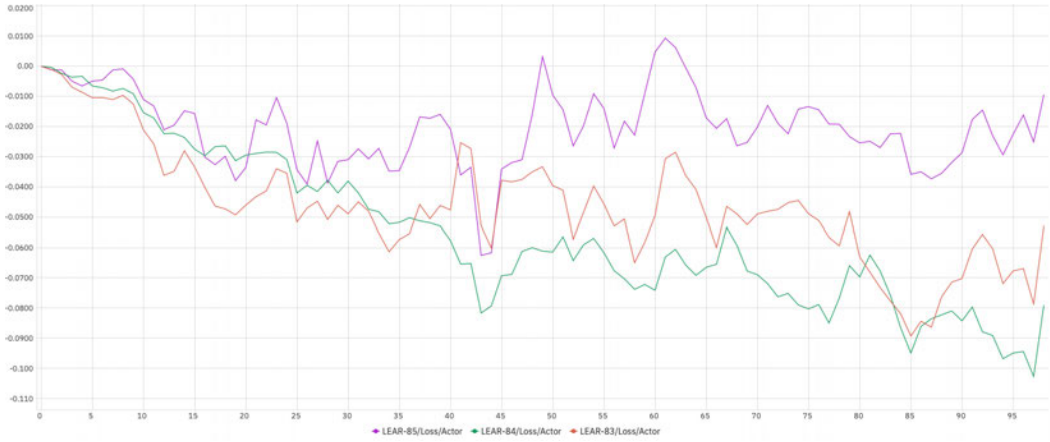

Figure 2: Actor losses for various reward signals. Purple: Number of nodes, orange: dual integral, green: primal integral.

Our ablation study revealed that selecting the negative change in the dual integral does not improve performance and, in some cases, results in degraded performance compared to other reward functions. This may be attributed to the environment's partial observability, where the number of nodes can describe the search tree in greater detail than the dual bound. We hypothesize that combining these three reward signals may improve solving time performance compared to other baselines.

6 Conclusion & Future Work

We have presented an approach to learning optimal variable selection for branching in B&B. Our method leverages an existing policy to receive informative feedback and has the potential to discover an improved policy. We have demonstrated on multiple datasets that our method outperforms SCIP's default solver but fails to do so when compared to current state-of-the-art behavioural cloning models.

We recommend continuing with our reinforcement learning model but suggest structuring the learning process similarly to approaches outlined in [15] or designing a reward signal using inverse reinforcement learning [14]. This would ensure that the agent effectively navigates the search space to outperform imitators. Furthermore, the architecture of our critic network could be enhanced by incorporating more advanced learning methods, such as graph attention mechanisms [22, 3, 10], to provide a more accurate representation of the value of the current state.

References

[1] BALAS, Egon ; HO, Andrew: Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. In: *Combinatorial Optimization* (1980), S. 37–60

[2] BERTSIMAS, Dimitris ; TSITSIKLIS, John N.: *Introduction to Linear Optimization*. Belmont, MA : Athena Scientific, 1997. – ISBN 9781886529199

[3] CANGEA, Cătălina ; VELIČKOVIĆ, Petar ; JOVANOVIĆ, Nikola ; KIPF, Thomas ; LIÒ, Pietro: *Towards Sparse Hierarchical Graph Classifiers*. 2018. – URL `https://arxiv.org/abs/1811.01287`

[4] CORNUÉJOLS, Gérard ; SRIDHARAN, Ranjani ; THIZY, Jean-Michel: A comparison of heuristics and relaxations for the capacitated plant location problem. In: *European journal of operational research* 50 (1991), Nr. 3, S. 280–297

[5] DEGRIS, Thomas ; WHITE, Martha ; SUTTON, Richard S.: *Off-Policy Actor-Critic*. 2013. – URL https://arxiv.org/abs/1205.4839

[6] GASSE, Maxime ; CAPPART, Quentin ; CHARFREITAG, Jonas ; CHARLIN, Laurent ; CHÉTELAT, Didier ; CHMIELA, Antonia ; DUMOUCHELLE, Justin ; GLEIXNER, Ambros ; KAZACHKOV, Aleksandr M. ; KHALIL, Elias ; LICHOCKI, Pawel ; LODI, Andrea ; LUBIN, Miles ; MADDISON, Chris J. ; MORRIS, Christopher ; PAPAGEORGIOU, Dimitri J. ; PARJADIS, Augustin ; POKUTTA, Sebastian ; PROUVOST, Antoine ; SCAVUZZO, Lara ; ZARPELLON, Giulia ; YANG, Linxin ; LAI, Sha ; WANG, Akang ; LUO, Xiaodong ; ZHOU, Xiang ; HUANG, Haohan ; SHAO, Shengcheng ; ZHU, Yuanming ; ZHANG, Dong ; QUAN, Tao ; CAO, Zixuan ; XU, Yang ; HUANG, Zhewei ; ZHOU, Shuchang ; BINBIN, Chen ; MINGGUI, He ; HAO, Hao ; ZHIYU, Zhang ; ZHIWU, An ; KUN, Mao: *The Machine Learning for Combinatorial Optimization Competition (ML4CO): Results and Insights*. 2022. – URL https://arxiv.org/abs/2203.02433

[7] GASSE, Maxime ; CHETELAT, Didier ; FERRONI, Nicola ; CHARLIN, Laurent ; LODI, Andrea: Exact Combinatorial Optimization with Graph Convolutional Neural Networks. In: WALLACH, H. (Hrsg.) ; LAROCHELLE, H. (Hrsg.) ; BEYGELZIMER, A. (Hrsg.) ; ALCHÉ-BUC, F. d'(Hrsg.) ; FOX, E. (Hrsg.) ; GARNETT, R. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 32, Curran Associates, Inc., 2019. – URL https://proceedings.neurips.cc/paper_files/paper/2019/file/d14c2267d848abeb81fd590f371d39bd-Paper.pdf

[8] HE, He ; DAUME III, Hal ; EISNER, Jason M.: Learning to Search in Branch and Bound Algorithms. In: GHAHRAMANI, Z. (Hrsg.) ; WELLING, M. (Hrsg.) ; CORTES, C. (Hrsg.) ; LAWRENCE, N. (Hrsg.) ; WEINBERGER, K.Q. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 27, Curran Associates, Inc., 2014. – URL https://proceedings.neurips.cc/paper_files/paper/2014/file/757f843a169cc678064d9530d12a1881-Paper.pdf

[9] LAND, Ailsa H. ; DOIG, Alison G.: An Automatic Method of Solving Discrete Programming Problems. In: *Econometrica* 28 (1960), S. 497. – URL https://api.semanticscholar.org/CorpusID:35442133

[10] LEE, Junhyun ; LEE, Inyeop ; KANG, Jaewoo: *Self-Attention Graph Pooling*. 2019. – URL https://arxiv.org/abs/1904.08082

[11] LILLICRAP, Timothy P. ; HUNT, Jonathan J. ; PRITZEL, Alexander ; HEESS, Nicolas ; EREZ, Tom ; TASSA, Yuval ; SILVER, David ; WIERSTRA, Daan: *Continuous control with deep reinforcement learning*. 2019. – URL https://arxiv.org/abs/1509.02971

[12] MAHER, Stephen ; MILTENBERGER, Matthias ; PEDROSO, Joã. P. ; REHFELDT, Daniel ; SCHWARZ, Robert ; SERRANO, Felipe: PySCIPOpt: Mathematical Programming in Python with the SCIP Optimization Suite. In: *Mathematical Software – ICMS 2016*. Springer International Publishing, 2016, S. 301–307

[13] MITTELMANN, Hans: *Decision Tree for Optimization Software*. – URL https://plato.asu.edu/bench.html

[14] NG, Andrew Y. ; RUSSELL, Stuart u. a.: Algorithms for inverse reinforcement learning. In: *Icml* Bd. 1, 2000, S. 2

[15] PARSONSON, Christopher W. F. ; LATERRE, Alexandre ; BARRETT, Thomas D.: *Reinforcement Learning for Branch-and-Bound Optimisation using Retrospective Trajectories*. 2022. – URL https://arxiv.org/abs/2205.14345

[16] PROUVOST, Antoine ; DUMOUCHELLE, Justin ; SCAVUZZO, Lara ; GASSE, Maxime ; CHÉTELAT, Didier ; LODI, Andrea: Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers. In: *Learning Meets Combinatorial Algorithms at NeurIPS2020*, URL https://openreview.net/forum?id=IVc9hqgibyB, 2020

[17] SCHULMAN, John ; MORITZ, Philipp ; LEVINE, Sergey ; JORDAN, Michael ; ABBEEL, Pieter: *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018. – URL https://arxiv.org/abs/1506.02438

[18] SCHULMAN, John ; WOLSKI, Filip ; DHARIWAL, Prafulla ; RADFORD, Alec ; KLIMOV, Oleg: *Proximal Policy Optimization Algorithms*. 2017. – URL https://arxiv.org/abs/1707.06347

[19] SILVER, Tom ; ALLEN, Kelsey ; TENENBAUM, Josh ; KAELBLING, Leslie: *Residual Policy Learning*. 2019. – URL https://arxiv.org/abs/1812.06298

[20] SMART, W.D. ; PACK KAELBLING, L.: Effective reinforcement learning for mobile robots. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)* Bd. 4, 2002, S. 3404–3410 vol.4

[21] TANG, Yunhao ; AGRAWAL, Shipra ; FAENZA, Yuri: *Reinforcement Learning for Integer Programming: Learning to Cut*. 2020. – URL https://arxiv.org/abs/1906.04859

[22] VELIČKOVIĆ, Petar ; CUCURULL, Guillem ; CASANOVA, Arantxa ; ROMERO, Adriana ; LIÒ, Pietro ; BENGIO, Yoshua: *Graph Attention Networks*. 2018. – URL https://arxiv.org/abs/1710.10903