# INTELLIGENT VARIABLE SELECTION FOR BRANCH & BOUND METHODS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Combinatorial optimisation Wolsey & Nemhauser (1999) is applied to a wide variety of real-world problems like job scheduling, capacity planning and supply-chain management. These problems are usually modelled as Mixed Integer Programming (MIP) Problems and solved using the Branch and Bound (B&B) Land & Doig (2010) paradigm. Branch and Bound method partitions the solution space (branching) by creating constrained sub-problems (bounding) and explores those subsets of the solution space which are highly likely to produce optimal solutions. The efficiency of the Branch and Bound method in finding optimal solutions is heavily influenced by the variable and node selection heuristics used for branching Linderoth & Savelsbergh (1999). In this paper, we propose a novel deep reinforcement learning based variable selection strategy. The proposed solution shows significant improvement over Strong Branching (SB) strategies Applegate et al. (1995), which have been traditionally used for variable selection. The solution also outperforms the current state of the art RL-based branching strategies like PPO Schulman et al. (2017) and DQN Hessel et al. (2018). The results of our experiments show that the proposed solution is robust and scalable to different kind of problems.

## 1 INTRODUCTION

Mixed Integer Programming (MIP) formulation is an essential step in solving real-world combinatorial optimisation problems. Real-world optimisation problems are complex in nature and are often NP-Hard due to the number of integer variables involved in these problems. The increase in the number of integer variables causes the task of finding the solution set that minimises the optimisation objective becomes tedious because while obeying the constraints to becomes tedious and computationally costly due to the expansion of the solution space with increase in the number of integer variables. Branch and Bound Land & Doig (2010) (B&B) method is a widely used strategy to solve Mixed Integer Programming problems.

Branch and Bound method performs an iterative tree search of the solution space by partitioning the solution space into disjoint subsets creating sub-problems. The search selects sub-problems in a fashion avoids exploration of subsets unlikely to produce optimal solutions and actively explores subsets of the solution space more likely to produce optimal solutions. Different branching strategies, pruning and cutting rules are used to search the solution space effectively to fasten the search to obtain optimal solutions.

Branching strategies are key to solving Branch and Bound problems as robust branching can result in reduction in the effective size of the solution space and obtain optimal solutions at a faster rate. Variable selection based branching Glankwamdee (2004) determines the next variable to be used for partitioning the subspace and node selection based branching determines the next subset of solution space to be explored. The focus of this paper will be on variable selection based branching.

Traditional strategies for variable selection like Psuedo-Cost Branching and Strong Branching Applegate et al. (1995) are manually designed using domain knowledge for each problem instance and do not generalise to a broad set of problems. Learning to branch using machine learning methods helps in generating variable selection strategies that are non-myopic and can generalise to a broader set of problems. Supervised Learning and Imitation Learning based approaches are a common choice for learning to branch and are trained using expert demonstrations. These methods face

issues when they are applied to real-world problems that differ from the expert demonstrations that they were trained on.

Reinforcement Learning has also been adopted for variable selection in the literature. However, it suffers from cold-start problems and cannot perform optimally until sufficient experience tuples are collected, resulting in sub-optimal performance. In this paper, a novel deep reinforcement learning based variable selection strategy is proposed where we make use of demonstration data initially during training to avoid cold-start problem. The key contributions of the paper are as follows,

- A deep reinforcement learning framework for branching variable selection has been developed which is used in Branch & Bound methods to solve Mixed Integer Programming (MIP) problems.
- The proposed solution uses prioritised experience replay, entropy regularized policy updates and double critics to enhance learning stability.
- The proposed solution uses demonstration data during beginning of training to avoid cold-start problem.
- The proposed solution uses MaxLP Gain construction heuristic to traverse tree and generates experience tuples to avoid partial observability.
- We assess the performance of our proposed strategy on heuristic based and SOTA RL based branching strategies and to different kind of MIP formulations and observe that our solution generalizes well to the different scenarios.

## 2 Related Work

**Traditional Variable Selection Strategies** Most Infeasible Branching (MIB) Achterberg et al. (2005), Pseudo-Cost Branching (PC) and Strong Branching (SB) Applegate et al. (1995) are traditional methods which score the candidate variables based on their effectiveness in partitioning the solution space. *Most Infeasible Branching* method chooses the variable with the high fractional parts (close to 0.5) and has been shown in Achterberg et al. (2005) to perform worse than if the variables were to be selected randomly . *Pseudo-cost Branching* Falk (1980) estimates an expected gain for each candidate variable based on the history of branchings. The method suffers from cold-start due to lack of history and requires the pseudo-costs to be initialised manually. *Strong Branching* is an efficient strategy which selects candidate variables that result in maximum improvement the dual bound at every stage. However, this strategy is computationally costly even for small problems and becomes intractable even for a slight increase in the size of the problem.

**Learning based Variable Selection Strategies** As traditional variable branching strategies are computationally heavy and require manual intervention, machine learning based strategies have been studied to develop strategies that are devoid of these drawbacks. Expert demonstrations of Strong Branching (SB) strategy are used to train *Supervised Learning* based techniques like offline extra tree regression Geurts et al. (2006) and online linear regression Marcos Alvarez et al. (2016) to predict branching scores and *Imitation Learning* based agents which learn to branch by mimicking expert behavior. However, these methods require extensively labelled expert datasets for training and cannot generalise well to heterogeneous problems. *Reinforcement Learning* has also been used to solve the variable branching problem, Sun et al. (2020) designs a primal-dual policy network with negative unit reward for every timestep to encourage the agent to learn to solve the problem with minimum number of steps. In Parsonson et al. (2022), Double Deep Q Network architecture with a Superior Network is used to deal with the large action spaces.

## 3 Background

### 3.1 Mixed Integer Programming (MIP)

**Defintion 1 Mixed Integer Programming (MIP) Formulation** Given A $\in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$, and a subset $I \subseteq \{1, 2, ..., n\}$, the mixed integer program $MIP = (A, b, c, I)$ linear in x is defined as follows,

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

where the vectors in $X$ are feasible solutions of the MIP if $X = \{x \in \mathbb{R}^n | Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z}, \forall j \in I\}$, and $x^* \in X$ is an optimal solution if $c^T x^* = z^*$. The problem is NP-Hard due to the integer requirements on x. MIP problems are solved by recursively splitting the problem into sub-problems using Branch and Bound method, where the LP relaxation of the problem is used to circumvent the integerality requirements and to obtain the lower bound for the sub-problems.

**Defintion 2 LP Relaxation for a (MIP) Problem** The LP relaxation of the MIP is defined as follows,

$$\check{z} = min\{c^T x | Ax \leq b, x \in \mathbb{R}^n\} \tag{2}$$

The lower-bound for the subtree is provided by solving the above LP-relaxation and if the objective value of the LP relaxation $\check{z} \leq \hat{z}$ where $\hat{z} = c^T \hat{x}$ of the current best solution $\hat{x}$, then the node can be discarded. The optimal integer solution will always lie between the determined upper-bound and lower-bound for the node to be explored. The choice of the variable $x_j$ to be used for calculating these bounds, the choice of the nodes to be explored and the cutting rules are different strategies used in solving MIP problems.

### 3.2 Branch & Bound (B&B) Algorithm

For any optimisation problem defined as $O = (W, f)$ with an objective function $f : W \rightarrow \mathbb{R}$ and $W$ as the search space or the set of all valid solutions to $O$, the branch and bound algorithm aims to find an optimal solution $x^* \in argmin_{x \in W} f(x)$. The search space $W$ is partitioned into disjoint sub-spaces by constructing a search tree $T$, which is then iteratively partitioned by selecting a subspace $G \subset W$ from the unexplored subspaces and checking for $\hat{x}' \in G$ has $f(\hat{x}') < f(\hat{x})$ for a feasible $\hat{x} \in W$ stored globally. If above condition is satisfied then the incumbent solution is updated and the subspace $G$ is branched into subspaces $G_1, G_2, ..., G_k$ and the search continues iteratively.If $f(\hat{x}') \geq f(\hat{x})\ \forall x \in G$, the subspace is abandoned and not searched further. Once the search is complete with no unexplored subspaces, the best incumbent solution is returned and the search terminates.

## 4 Proposed Solution

Variable Selection is crucial in Branch and Bound algorithm as it prescribes the variable on which branching is to be performed. Selecting a wrong variable for branching can result in increase in the size of the search tree and can increase the computational complexity significantly. Existing methods score the different variables and perform branching greedily over the variable that has the lowest score. However, the need for high-quality variable selection requires a strategy that is far-sighted and is capable of generalising to unseen expert data. Reinforcement Learning is leveraged in the proposed solution to achieve above goals by modelling the variable selection process as a Markov Decision Process (MDP) Puterman (2014).

### 4.1 MDP Formulation

The sequential decision making process for variable selection can be formulated as a Markov Decision Process. With the solver as the environment, the brancher agent state space $S$, action space $A$, transition function $P$ and the reward function $R$ are described as follows,

**State Space** $S$ At each instance $t$, the problem state $s_t$ consists of the bi-partite graph representation for the MIP of the focus node and the index set $I_t$ of the branching candidates which posses integer constraints and have a non-integer solution. The state $s_t$ is defined as below,

$$s_t = \{(X_t, E_t, C_t), I_t\} \tag{3}$$

where $X \in \mathbb{R}^{n \times d_x}$ is a feature matrix describing the variables and the features from the objective function and the constraints. The nodes of the bi-partite graph correspond to the $n$ variables to be optimised and the $m$ constraints to be met. An edge $e_{ij}$ is created if the variable $x_i$ has a coefficient $A_{ij} \neq 0$ in the constraint $c_j$ with features $d_e$ representing the constraint's constant term resulting in $E \in \mathbb{R}^{m \times n \times d_e}$, the edge feature matrix. $C \in \mathbb{R}^{m \times d_c}$ represents the constraint matrix where the constraints are encoded into $d_c$ features like in Gasse et al. (2019). Thus, the state $s_t$ presents

a holistic representation of the variables, the objective function and the constraints involved in the MIP at any instance $t$.

**Action Space** $A$ The action $a_t$ represents the branching variable selected at instance $t$ from the action space where $I_t$ is the index set of branching variable candidates.

$$A(s_t) = \{i \in I_t\} \tag{4}$$

**Transition Function** $P$ The state transition function $P(s_{t+1}|s_t, a_t)$ generates the next state provided the current state and action. The next state $s_{t+1}$ is given by using a node selection policy $\pi_n$ which selects the next node to be branched upon, whose features will be represented by $s_{t+1}$. However, the environment changes independently of the action of the brancher agent as it cannot perceive the actions of the node selection policy resulting in partial observability. To circumvent partial observability, multiple trajectory paths are constructed such that the root-leaf pairs are used as the source-destination pair and each node on the search tree is mapped to a single trajectory. This generated trajectory information is then stored in the replay buffer and used during the training process.

**Reward Function** $R$ The goal of our solution is to be able to obtain an optimal solution faster with minimum branching steps as possible. Thus, we define the reward $R_t$ at instance $t$ as follows,

$$R_t = -1 \tag{5}$$

where for every step a unit penalty is issued. The cumulative reward of an episode will represent the number of steps to optimal solution as a penalty, thereby encouraging the method to solve in minimum number of steps.

### 4.2 SOLUTION

A novel deep reinforcement learning algorithm for variable branching based on Soft Actor Critic Networks Christodoulou (2019) with Prioritised Experience Replay Schaul et al. (2015) is proposed. The soft actor critic algorithm aims to learn an optimal actor $\pi^*$ which maximises the expected discounted reward and the entropy $H$ associated with it.

$$\pi^* = argmax_\pi \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)} \left[\gamma^t r(s_t, a_t) + \alpha H(\pi(.|s_t))\right] \tag{6}$$

where $H(\pi(.|s_t)) = -\log \pi(.|s_t)$, $\alpha$ is the temperature parameter that manages the exploration-exploitation trade off. The actor network and the critic network are modelled using Graph Convolutional Neural Network (GCNN) as the state $s_t$ is represented as a bipartite graph. Two successive passes to perform a single convolution as below,

$$c'_p \leftarrow f_c \left( c_p, \sum_{q}^{(p,q)\in E} g_c(x_q, e_{pq}, c_p) \right), \tag{7}$$

$$x'_q \leftarrow f_x \left( x_q, \sum_{p}^{(p,q)\in E} g_c(x_q, e_{pq}, c'_p) \right), \tag{8}$$

$\forall p \in C, \forall q \in X$, followed by a double-layer perceptrons with ReLU activations. The actor network uses masked softmax to estimate the action probability distribution $\pi$ and the critic network uses the masked sum to predict the state-action value $Q$. The objective function for the actor and critic networks are as follows,

$$J_\pi(\phi) = \mathbb{E}_{s_t} \left[\pi_t(s_t)^T[-\alpha \log \pi_t(s_t) - Q_\theta(s_t)]\right] \tag{9}$$

$$J_Q(\theta) = \mathbb{E}_{s_t,a_t} \left[Q_\theta(s_t, a_t) - [r(s_t, a_t) + \gamma[Q'_\theta(s_{t+1}, a_{t+1}) - \alpha \log \pi_\theta(a_{t+1}|s_t + 1)]]^2\right. \tag{10}$$

The action generated by the soft-actor agent $a_t$ represents the branching variable and the next node $n_t$ to be branched at instance $t$ is selected by a node selection policy. The sub-tree with $n_t$ as the root

node is traversed by selecting the leaf with the largest LP gain until a leaf node not yet fathomed is reached. Trajectories are generated for each of the nodes present in the sub-tree and are added to the experience replay buffer. The node selection policy is used when the soft actor-critic agent interacts with the environment and generates a variable for branching during both training and inference phases. However, the maximum LP-gain trajectory generation happens only during training phase having no additional inference-time overhead.

During the initial training phases, the brancher agent is naive due to lack of experience causing random actions to be taken, which can result in initial sub-optimal performance. The experience buffer filled with expert demonstration data at the beginning of training process will help close the performance gap during this time. The proposed solution is provided with tuples of expert demonstration data $\{s_t, a_t, r_t, s_{t+1}, done_t\}$ in the replay buffer during the beginning of training which is gradually phased it out in a linear matter as new experience tuples are collected by the brancher agent, to avoid cold-start problem during beginning of training.

**Algorithm 1:** SAC For Variable Selection for Branching

**for** *episode = 1 to E* **do**
  Initialize experience replay buffer $D$ with $b$ batch filled with demonstration tuples;
  Initialize online and target network weights $\theta$ and $\theta'$;
  Obtain initial observation from environment $s_0$;
  **for** *t = 1 to T* **do**
    Select action $a_t \sim \pi_t heta(.|s_t)$ Observe next state $s_{t+1}, a_t, r_t, s_{t+1}, done_t$;
    Store ($s_t$, $a_t$, $s_{t+1}$, $r_t$) in $D$;
    Set $s_t$ = $s_{t+1}$ using node selection policy which gives next node $n_t$;
    Explore $Traj \leftarrow$ MaxLPGain($s_t, a_t, r_t, s_{t+1}, n_t$);
    Store $Traj$ in $D$;
    **for** $i$ *= 1 to* $M$ *updates* **do**
      Sample a random batch of $S$ samples ($s_t$, $a_t$, $s_{t+1}$, $r_t$) from $D$;
      Set $y_t = r_t + \gamma \min_{j=1,2} Q^{\pi'}_{t,j}(s_t, a_t) - \alpha \log \pi_\theta(a_{t+1|s_{t+1}})$;
      Minimize critic loss for both $j = 1, 2$ $L(\theta_i) = \frac{1}{S}\sum_S (Q'_{t,j}(s_{t+1}, a_t) - y_t)^2$;
      **if** $t\%d$ *== 0* **then**
        Update policy $\mu_i$ with gradient
        $\nabla_{\theta_i} J(\mu_t) = \nabla_{\theta_i}\mu_t(a_t|o_t)\nabla_{a_t} Q_{1,t}(s_t, a_t) - \alpha \log \pi_\theta(a_{t+1|s_{t+1}})$;
        Update target critic network parameters as $\theta_{Q'_j} = \tau\theta_{Q_j} + (1-\tau)\theta_{Q'_j}$;
        Update target actor network parameters as $\theta_{\pi'} = \tau\theta_\pi + (1-\tau)\theta_{\pi'}$;
      **end**
    **end**
  **end**
**end**

## 5 Results and Discussion

In this section, we discuss results about the performance of our proposed control strategy and examine them for insights and improvement areas. We study the performance of our algorithm under two MIP settings and compare the performance of our algorithm with state of the art reinforcement learning methods used for branching. Second, we evaluate the performance of our control strategy by using metrics of different levels of control complexity as follows:

1. Non-Adaptive control – Describes the class of control strategies that are static and predefined. We compare the performance of our method opposed to heuristic based branching method Strong Branching (SB) Method;
2. Adaptive Control strategies - that act based on recommendations from SOTA RL controllers, here we study PPO & DQN, and compare its performance with the performance of our method which leverages SAC.

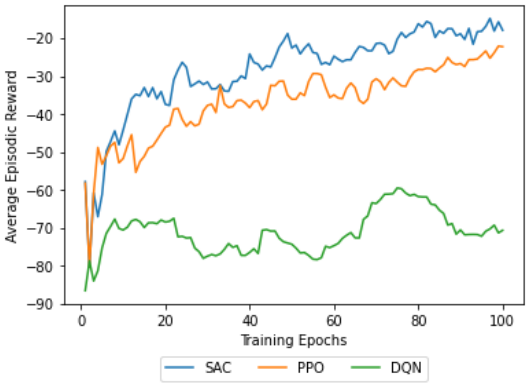


Figure 1: Comparison for MIP Setting 1

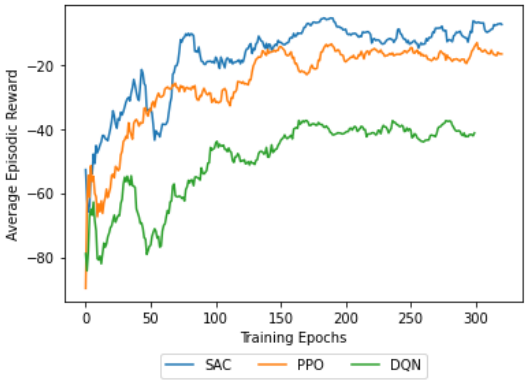


Figure 2: Comparison for MIP Setting 2

Table 1: Performance Metrics separated by Algorithms

| **MIP Setting 1** | | | | |
|---|---|---|---|---|
| **Metric** | **SAC** | **PPO** | **DQN** | **SB** |
| Average Reward | -19.72 | -27.21 | -68.34 | −− |
| Average # Nodes | 7.35 | 10.48 | 12.61 | 6.43 |
| Average # LPs | 198 | 238 | 262 | 170 |
| **MIP Setting 2** | | | | |
| **Metrics** | **SAC** | **PPO** | **DQN** | **SB** |
| Average Reward | -14.61 | -18.34 | -43.15 | −− |
| Average # Nodes | 4.27 | 4.58 | 5.68 | 3.76 |
| Average # LPs | 18.7 | 21.3 | 25.3 | 19.6 |

We used the open-source Ecole Prouvost et al. (2021) and PySCIPOpt Maher et al. (2016) libraries with SCIP 7.0.1 (SCIP) as the backend solver to do instance generation and testing. In total, we considered two NP-hard problem benchmarks: set covering and maximum independent set, along with SB branching strategy in addition to RL based controllers SAC (proposed), PPO and DQN. We have ommited direct comparison to the SOTA tuned commercial solvers, which we do not claim to be competitive with at this stage.

The PPO and DQN agents have the similar MDP formulation and reward definition so that they can be compared effectively. The node selection policy used for them is DFS search and we can see that DFS search is causing the algorithm to perform poorly as the DFS will become computationally heavy for practical MIPs. We compare our method's performance against the aforementioned algorithms on the different MIP settings a) Set Covering Balas & Ho (1980) referred to as MIP problem 1 and b) Maximum Independent Set Bergman et al. (2016) referred to as MIP problem 2. The average reward of our strategy improves over PPO by 27% and 22% in the MIP setting 1 and MIP setting 2 respectively. It also shows a significant improvement over DQN control by 70% and 58% for the MIP setting 1 and MIP setting 2 respectively. The proposed strategy exhibits steady learning behavior in the initial episodes unlike the PPO and DQN based branching strategies, this can be attributed to the expert demonstration data provided to our solution to avoid cold-start.

The behavior of the different algorithms in the MIP problem 2 are in-line with the performance of the algorithms on the MIP problem 1. It can see that the episodic reward for SAC in the MIP problem 2 starts initally at lower levels than that of the PPO and DQN. However, it exhibits steady episode-over-episode learning behavior until around 100 episodes, after which the agent's learning stabilises. DQN exhibits very slow learning as it ascends with much lower slope and takes until around 160 episodes to have stabilised learning. The efficiency of our framework in solving the CO problems can be inferred from the average # of nodes and the average # of LPs in Table. The proposed strategy is able to reduce the average # of nodes and average # of LPs significantly. SAC reduces the average # of nodes over PPO by 30% and 16% over the MIP problem 1 and MIP problem 2 respectively. It also shows significant impact in the average # of nodes over DQN by around 40%

and 20% for the different problems. The average # of LPs exhibit a decreasing trend across the different problem settings. Also, it performs as good as the SB control significantly across the different MIP problems. Thus, the improvement trend of our proposed strategy over PPO and DQN control is consistent across the different MIP formulations, elucidating that our method is robust and scalable to different kind of environments and significantly outperforms state of the art control strategies.

## 6 CONCLUSION

In this paper, we proposed variable selection strategy for Branch & Bound Methods leveraging Soft Actor Critic with Prioritized Experience Replay. Our method is capable of optimally performing variable selection for different kinds of Mixed Integer Programming problems as it is supplemented by demonstration data to avoid cold-start and trajectory generation that alleviates partial observability and increases sample efficiency. The proposed method outperforms existing rule-based and reinforcement learning based control strategies like Proximal Policy Optimization (PPO) and Deep Q Networks (DQN) and the traditional variable selection method Strong Branching control (SB) across different MIP problems. The proposed method was scalable to different problems and was robust compared to different kind of branching strategies.

## REFERENCES

Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

D. Applegate, R. Bixby, V. Chvatal, and B. Cook. Finding cuts in the tsp (a preliminary report). Technical report, 1995.

Egon Balas and Andrew Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. In *Combinatorial Optimization*, pp. 37–60. Springer, 1980.

David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John Hooker. *Decision diagrams for optimization*, volume 1. Springer, 2016.

Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

Patrick G Falk. Experiments in mixed integer linear programming in a manufacturing system. *Omega*, 8(4):473–484, 1980.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *CoRR*, abs/1906.01629, 2019. URL `http://arxiv.org/abs/1906.01629`.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

Wasu Glankwamdee. *Lookahead branching for mixed integer programming*. PhD thesis, Lehigh University, 2004.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Ailsa H Land and Alison G Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pp. 105–132. Springer, 2010.

Jeff T Linderoth and Martin WP Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

Stephen Maher, Matthias Miltenberger, João Pedro Pedroso, Daniel Rehfeldt, Robert Schwarz, and Felipe Serrano. PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In *Mathematical Software – ICMS 2016*, pp. 301–307. Springer International Publishing, 2016. doi: 10.1007/978-3-319-42432-3_37.

Alejandro Marcos Alvarez, Louis Wehenkel, and Quentin Louveaux. Online learning for strong branching approximation in branch-and-bound. 2016.

Christopher WF Parsonson, Alexandre Laterre, and Thomas D Barrett. Reinforcement learning for branch-and-bound optimisation using retrospective trajectories. *arXiv preprint arXiv:2205.14345*, 2022.

Antoine Prouvost, Justin Dumouchelle, Maxime Gasse, Didier Ch'etelat, and Andrea Lodi. Ecole: A library for learning inside milp solvers. *ArXiv*, abs/2104.02828, 2021.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Haoran Sun, Wenbo Chen, Hui Li, and Le Song. Improving learning to branch via reinforcement learning. 2020.

Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.