Reinforcement Learning for Variable Selection in a Branch and Bound Algorithm

Marc Etheve[1,3]([✉]) , Zacharie Alès[2,3] , Côme Bissuel[1] , Olivier Juan[1] , and Safia Kedad-Sidhoum[3]

[1] EDF R&D, Paris, France
{marc.etheve,come.bissuel,olivier.juan}@edf.fr
[2] ENSTA Paris, Institut Polytechnique de Paris, Paris, France
zacharie.ales@ensta-paris.fr
[3] CNAM Paris, CEDRIC, Paris, France
safia.kedad_sidhoum@cnam.fr

Abstract. Mixed integer linear programs are commonly solved by Branch and Bound algorithms. A key factor of the efficiency of the most successful commercial solvers is their fine-tuned heuristics. In this paper, we leverage patterns in real-world instances to learn from scratch a new branching strategy optimised for a given problem and compare it with a commercial solver. We propose FMSTS, a novel Reinforcement Learning approach specifically designed for this task. The strength of our method lies in the consistency between a local value function and a global metric of interest. In addition, we provide insights for adapting known RL techniques to the Branch and Bound setting, and present a new neural network architecture inspired from the literature. To our knowledge, it is the first time Reinforcement Learning has been used to fully optimise the branching strategy. Computational experiments show that our method is appropriate and able to generalise well to new instances.

Keywords: Reinforcement learning · Mixed integer linear programming · Neural network · Branch and bound · Branching strategy

1 Introduction

Mixed Integer Linear Programming (MILP) is an active field of research due to its tremendous usefulness in real-world applications. The most common method designed to solve MILP problems is the Branch and Bound (B&B) algorithm (see [1] for an exhaustive introduction). B&B is a general purpose procedure dedicated to solve any MILP instance, based on a divide and conquer strategy and driven by generic heuristics and bounding procedures.

Recently, a lot of attention has been paid to the interactions between MILP and machine learning. As pointed out in [2], learning methods may compensate

for the lack of mathematical understanding of the B&B method and its variants [3,4]. The plethora of different approaches in this young field of research gives evidence of the variety of ways in which learning can be leveraged. For instance, a natural idea is to bypass the whole B&B procedure to directly learn solutions of MILP instances [5]. If one wants to preserve the optimality guarantee provided by the B&B algorithm, a solution could be to rather learn the output of a computationally expensive heuristic used in a B&B scheme [6–8]. Alternatively, [9] suggests learning to select the best cut among a set of available cuts at each node of the B&B tree. Whether it is by Imitation Learning or by Reinforcement Learning (RL), these solutions are often limited by their scope: they seek to take decisions according to a local criterion.

In the present work, we propose FMSTS (*Fitting for Minimising the SubTree Size*), a novel approach based on Reinforcement Learning aiming at optimising a global criterion at the scale of the whole B&B tree. We learn a branching strategy from scratch, independent of any heuristic.

The paper is structured as follows. First, we define the general setting of our study. Using RL to minimise a global criterion, we then demonstrate that, under certain assumptions, a specific kind of value functions enforces the optimality of such criterion. Next, we propose to adapt known generic learning methods and neural network architectures to the Branch and Bound setting. We illustrate our proposed method on industrial problems and discuss it before concluding.

2   General Setting

In real-world applications, companies often optimise systems on a regular basis given fluctuating data. This case has been studied in the literature for different purposes, such as learning an approximate solution [5] or imitating heuristics [8]. However, to our knowledge, no concrete contribution has been made regarding the use of Reinforcement Learning for variable selection (branching) in this setting. The present work fills this gap.

Throughout this paper, we are interested in the following setting. For a given problem $\mathcal{P}$, the instances are perceived as randomly distributed according to an unknown distribution $\mathcal{D}$. This distribution, emanating from real-world systems, governs the fluctuating data (A, b, c) across instances, written as

$$p \in \mathcal{P} : \begin{cases} \min_{x \in \mathbb{R}^n} c^\top x \\ \quad\quad Ax \leq b \\ s.t. \\ \quad\quad x_{\mathcal{J}} \in \{0, 1\}^{|\mathcal{J}|}, \ x_{-\mathcal{J}} \in \mathbb{R}^{n-|\mathcal{J}|} \end{cases} \tag{1}$$

with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. In practice, as the instances come from a single problem, they share the same structure. Especially, the set of null coefficients, the number of constraints m, of variables n and the set of binary variables $\mathcal{J}$ are the same for every instance of a given problem.

In this setting, we seek to learn and optimise a branching strategy to solve to optimality any instance of a given problem. As pointed out in [10], the problem of optimising the decisions along a B&B tree is naturally formulated as a control problem on a sequential decision-making process. More specifically, it is equivalent to solving a finite-horizon deterministic Markov Decision Process (MDP) and may thus be tackled by Reinforcement Learning (see [11] for an introduction).

3   Fitting for Minimising the SubTree Size (FMSTS)

In a finite-horizon setting, Reinforcement Learning aims at optimising an agent to produce sequences of actions that achieve a global objective. The agent is guided by local costs associated with the actions it takes. Starting from an initial state in a possibly stochastic environment, it must learn to take the best sequence of actions and thus transitioning from state to state to minimise the overall costs. Exact Q-learning solves such problem by updating a table mapping (Q-function) from state/action pairs to discounted future costs (Q-values).

In the following, we define the RL problem of interest and, as exact Q-learning is not practicable, the approximate framework considered. Next, we propose a specific informative Q-function allowing us to use this framework in practice.

3.1   Approximate Q-Learning with Observable Q-Values

Let us denote by $\mathcal{S}$ the set of every reachable state for a given problem $\mathcal{P}$, a state being defined as all the information available when taking a branching decision in a B&B tree. Under perfect information, a state associated to a B&B node is the whole B&B tree as it has been expanded at the time the branching decision is taken. We write $\mathcal{A}$ the set of actions, *i.e.* the set of available branching decisions on a specific problem (the set of binary variables $\mathcal{A} = \mathcal{J}$ in our case) and π a policy mapping any state to a branching decision:

$$\pi \ : \ \begin{cases} \mathcal{S} \to \mathcal{A} \\ s \mapsto \pi(s) = a \end{cases}.$$

The transitions between states are governed by the B&B solver, and the MDP is regarded as deterministic: given an instance and a state, performing an action will always lead to the same next state. In practice, such assumption is met as soon as the solver's decisions (apart from branching) are non stochastic.

We call agent any generator Π of branching sequences following policy π and denote $\Pi(p)$ the sequence of decisions that maps an instance p to a complete B&B tree. Let $\mu\left(\Pi(p)\right)$ be any metric of interest on the tree generated by Π for an instance p, and assume this metric is to be minimised. For instance, we can think of μ as the size of the generated tree, the number of simplex iterations, etc. In this setting, we are looking for the $\mu-$optimal agent Π^* such that

$$\Pi^* \in \arg\min_{\Pi} \mathbb{E}_{p\sim\mathcal{D}}\left[\mu\left(\Pi\left(p\right)\right)\right]. \tag{2}$$

Note here that the expectation is only on the MILP instances, as the MDP is deterministic.

Let us assume that one can define a Q-function Q^π which is consistent with μ, in the sense that μ is minimised if $\pi(s) = \arg\min_{a \in \mathcal{A}} Q^\pi(s, a)$. Even in this case, exact Q-learning cannot be used to minimise μ. First, maintaining an exact table for the Q-function is not tractable due to the size of $\mathcal{S}$, including for small real-world problems. Second, the transition from a state to the next is too complex to be modeled since it partly results from a linear optimisation.

To bypass these problems, we approximate the Q-function (see [11] for an introduction to Approximate Q-learning) by a neural network $\hat{Q}$ parametrised by θ and optimised by a dedicated gradient method as in the DQN (Deep Q-Network) approach [12]. We define the policy π_θ resulting from the Q-network as $\pi_\theta(s) = \arg\min_{a \in \mathcal{A}} \hat{Q}(s, a; \theta)$.

When facing a deterministic MDP, the exact Q-value $Q^{\pi_\theta}(s, a)$ of an action a given a state s and a policy π_θ is not stochastic and thus may be observable. In that case, the classic Temporal Difference loss used in [12] for training the Q-Network comes down to the simple expression

$$L(\theta) = \mathbb{E}_{s,a \sim \rho(.)} \left[\left(Q^{\pi_\theta}(s, a) - \hat{Q}(s, a; \theta) \right)^2 \right] \tag{3}$$

where ρ is the behaviour distribution of our agent, as stated in [12]. Note that, in Eq. (3), the observed Q-values are naturally influenced by parameter θ through the policy. Such loss is actually intuitive: if Q^{π_θ} is consistent with μ, if each action has non-zero probability to be taken in any encountered state and if $L(\theta) = 0$, then each B&B tree built by agent Π_θ (following π_θ) is optimal with respect to μ with probability 1.

3.2   Using the Subtree Size as Value Function

As highlighted in [8], an important difficulty when applying Reinforcement Learning to B&B algorithms is the credit assignment problem [13]: in order to determine the actions that lead to a specific outcome, one may define non-sparse informative local costs (negative rewards) consistent with the global objective.

This is not mandatory in a RL setting but may facilitate the learning task.

We choose the number of nodes in the generated tree as the global metric μ. This metric is often used to compare B&B methods (see for instance [6–8]), as it is a proxy for computational efficiency and independent of hardware considerations.

One of the main contributions of this paper is to propose a local Q-function Q^π which is consistent with the chosen global metric μ. We take advantage of the deterministic aspect of the environment and define $Q^\pi(s, a)$ as the size of the subtree rooted in the B&B node corresponding to s generated by action a and policy π. As stated in Proposition 1, this particular Q-function is not consistent in general with our choice of global criterion μ. Nonetheless, Proposition 2 asserts its optimality when using Depth First Search as node selection strategy.

Proposition 1. *In general, minimising the size of the subtree under any node in a B&B tree is not optimal with respect to the tree size.*

The proof is omitted for the sake of conciseness, but one can prove that minimising the subtree size can be sub-optimal when using Breadth First Search as the node selection strategy.

Proposition 2. *When using Depth First Search (DFS) as the node selection strategy, minimising the whole B&B tree size is achieved when any subtree is of minimal size.*

Proof. Let us call $\mathcal{O}$ the set of open nodes at a given iteration of the B&B process for a specific instance. The set of closed nodes (either by pruning or branching) is denoted $\mathcal{C}$.

We write $V^\pi (s \,|\, \zeta, \eta)$ the size of the subtree under s, entirely determined by the policy π followed in this subtree, a set of primal bounds ζ found in other subtrees and a node selection strategy η. When using DFS, the subtrees under each open node are expanded and fully solved sequentially, thus we can assume with no loss of generality that $\mathcal{O}$ is equal to $\{1, ..., k\}$ and is sorted according to the planned visiting order. In that case, the size V of the whole B&B tree can be expressed as

$$V = |\mathcal{C}| + \sum_{i=1}^{k} V^{\pi_i} \left(s_i \,\left|\, \{z_0\} \cup \left(\bigcup_{j<i} \zeta_j \right) , \ \eta = DFS \right. \right)$$

with ζ_i the set of all bounds to be found in the subtree rooted in s_i and z_0 the best bound obtained in $\mathcal{C}$.

It remains to prove that π_1 is optimal only if it leads to the minimal subtree under s_1. As two separate subtrees can only affect each other through their best primal bound under DFS, we have

$$V = |\mathcal{C}| + V^{\pi_1} (s_1 \,|\, \{z_0\}, \ \eta = DFS) + \sum_{i=2}^{k} V^{\pi_i} (s_i \,|\, \{z_{i-1}\}, \ \eta = DFS)$$

with $z_{i-1} = \min \left\{ \{z_0\} \cup \left(\bigcup_{j<i} \zeta_j \right) \right\}$.

Since the B&B procedure (with a gap set to zero) guarantees that we find the best primal bound of any expanded subtree, $(z_{i \, i=1})^k$ are completely independent of the branching policies, which gives, for any π_j, $j \in \{2, ..., k\}$:

$$\operatorname*{arg\,min}_{\pi_1 \in \Pi_1} V = \operatorname*{arg\,min}_{\pi_1 \in \Pi_1} V^{\pi_1} (s_1 \,|\, \{z_0\}, \ \eta = DFS)$$

with Π_1 the set of all valid branching policies under s_1. Therefore, choosing any other policy than $\pi_1 \in \operatorname*{arg\,min}_{\pi_1 \in \Pi_1} V^\pi (s_1 \,|\, \{z_0\}, \ \eta = DFS)$ is sub-optimal with respect to the tree size. $\qquad\square$

In the remaining, we use DFS as the node selection strategy according to Proposition 2. We now focus on optimising the branching strategy (variable selection) to minimise at each node the size of the underlying subtree. If we write $D_0^{\pi(s)}(s)$ and $D_1^{\pi(s)}(s)$ the child nodes of s following policy π, such value function satisfies the Bellman Equation (4). The relationship between the value and the Q-function is trivially defined by $Q^\pi(s,a) = 1 + V^\pi(D_0^a(s)) + V^\pi(D_1^a(s))$.

$$V^\pi(s) = 1 + V^\pi\left(D_0^{\pi(s)}(s)\right) + V^\pi\left(D_1^{\pi(s)}(s)\right) \tag{4}$$

This value function has two advantages. First, it is observable as assumed earlier: we only need to count the number of inheriting nodes once the B&B tree is fully expanded. Second, it is a local objective which guarantees the optimality of a global criterion, hence allowing us to perform RL without designing a suboptimal reward using any domain knowledge.

3.3 Algorithm

Using Approximate Q-learning and the subtree size as value function leads us to propose the FMSTS algorithm (Algorithm 1). Using Experience Replay and ε-greedy exploration as in [12], the algorithm essentially boils down to consecutively solving a MILP instance following the current policy or random choices with probability ε, fitting the observed values sampled from an experience replay buffer and iterating with the updated policy.

Algorithm 1. FMSTS

    for t = 0,...,N-1 **do**

        Draw randomly an instance p.

        Solve p following π_{θ_t} with ε-greedy exploration.

        Collect experiences along the generated tree $\left(s_i, a_i, Q^{\pi_{\theta_t}}(s,a), \hat{Q}(s,a;\theta_t)\right)$ and store them into an experience replay buffer $\mathcal{B}$.

        Update to θ_{t+1} using loss (3) on experiences drawn from $\mathcal{B}$.

    end for

---

4 Adapting Learning to the Branch and Bound Setting

To ensure the success of the FMSTS method (Algorithm 1) with respect to the objective (2), we need to adapt some components to the Branch and Bound setting. First, we adapt the loss guiding the neural network's training. Next, we use Prioritized Experience Replay while normalising probabilities. Last, we propose a new neural network architecture inspired from the literature.

4.1   Minimising an Expectation on the Instance Distribution

The loss defined by Eq. (3) does not seem to correspond to our objective (2). Indeed, it naturally gives more importance to the biggest trees, which can be heavily instance dependent. To neutralise this effect, we weight the loss by the inverse of the size of the corresponding B&B tree generated by the agent:

$$L(\theta) = \mathbb{E}_{s,a \sim \rho(.)} \left[\frac{1}{V^{\pi_\theta}(r(s))} \left(Q^{\pi_\theta}(s,a) - \hat{Q}(s,a;\theta) \right)^2 \right] \tag{5}$$

with $r(s)$ the root node of the tree containing s, such that $V^{\pi_\theta}(r(s))$ corresponds to the size of this tree. Then, any instance has equal weight in loss (5).

4.2   Performing Prioritized Experience Replay

Prioritized Experience Replay [14] biases the uniform replay sampling of Experience Replay [12] towards experiences with high Temporal Difference errors, *i.e.* when the predicted Q-values are far from their target. In FMSTS, an experience is a 4-tuple $\left(s_j, a_j, Q^{\pi_{\theta_j}}(s_j, a_j), \hat{Q}(s_j, a_j; \theta_j) \right)$ and the target $Q^{\pi_{\theta_j}}(s_j, a_j)$ is observed, which reduces the error to the simple expression $|Q^{\pi_{\theta_j}}(s_j, a_j) - \hat{Q}(s_j, a_j; \theta_j)|$.

In the context of sampling experiences in a B&B tree, one should take into account that the scale of the target $Q^{\pi_{\theta_j}}$ may vary exponentially both along the tree and across instances. As the scale of the error may likely vary with that of the target, we normalise this error by the target to get the sampling probability in the experience replay buffer

$$p_j \propto \frac{|Q^{\pi_{\theta_j}}(s_j, a_j) - \hat{Q}(s_j, a_j; \theta_j)|}{Q^{\pi_{\theta_j}}(s_j, a_j)}. \tag{6}$$

4.3   Designing a Regressor for the Q-Function

As in [6], we use both static and dynamic features to represent a state. Although many features may be relevant for the states' encoding, we opted to keep them limited in the present work. For static information, we perform a dimension reduction by PCA [15]: each instance is represented as the concatenation of its data (A, b, c) and PCA is applied on the resulting vectors. Our representation also includes the following dynamic features: the node's depth, the distance of the current primal solution to the bounds and the branching state. Concretely, the branching state B is one-hot encoded in a $3|\mathcal{J}|$ vector. Let us call $\mathcal{B}_0$ and $\mathcal{B}_1$ the set of variables that have been respectively set to 0 and 1 in the ascendant nodes of the current state. With no loss of generality, let us assume that $\mathcal{J} = \{1, ..., J\}$. Then we have $B_j = \mathbb{1}_{x_j \in \mathcal{B}_0}$, $B_{j+J} = \mathbb{1}_{x_j \in \mathcal{B}_1}$ and $B_{j+2J} = 1 - B_j - B_{j+J}$ for any $j \in \mathcal{J}$.

The chosen Q-function is essentially multiplicative, in the sense that the ratio between the targets in two consecutive states may be of magnitude 2 due to the

binary tree structure. In addition, the scale of these targets may strongly vary between instances. A basic feedforward neural network, based on summations, may struggle to handle such phenomena. To compensate for these effects and adapt to the B&B setting, we take inspiration from the Dueling architecture of [16] and propose the Multiplicative Dueling Architecture (MDA). As shown in Fig. 1, MDA implements the product between the 1-D output of a block of fully-connected layers fed with static features and the $|\mathcal{J}|$-D output of a block fed with both static and dynamic features. A linear activation on the 1-D output allows our agent to capture the variability of the chosen Q-function.
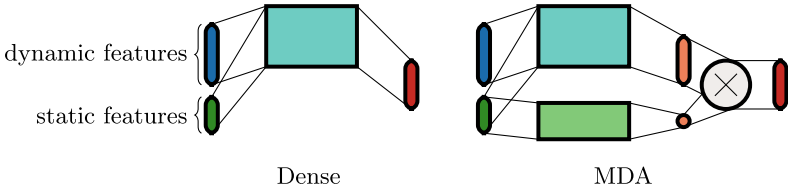

Fig. 1. Dense and Multiplicative Dueling architectures for the Q-network. The rectangles represent consecutive dense layers, the lightblue block being fed with all the features whereas the lightgreen one is only fed with static features (darkgreen). The output of the MDA is the product between a single unit and a $|\mathcal{J}|$-unit dense layer. (Colour figure online)

5   Experiments and Discussion

We test our algorithms on two sets of instances provided by Electricité de France (EDF), a french electric utility company. They are drawn from two different problems, one is related to energy management in a microgrid ($\mathcal{P}_1$) whereas the other one comes from a hydroelectric valley ($\mathcal{P}_2$). The problems have respectively 186 and 282 constraints, 120 and 207 variables, and 54 and 96 binary variables.

We compare our algorithms to the default branching strategy of CPLEX (denoted CPLEX in the following) and full Strong Branching (denoted SB). We use CPLEX 12.7.1 [17] under DFS while turning off all presolving and cutting.

To avoid any dependency of our results to the train or the test set, we present cross-validated results. Algorithm 1 is run 100 times independently on randomly partitioned train and test sets. Each time, 200 instances are used for training while 500 unseen instances are used for testing. Figure 2 shows the averaged number of nodes in the complete B&B trees on the test sets during the learning process: test instances are solved using the strategy learned on train instances at the current iteration of Algorithm 1.
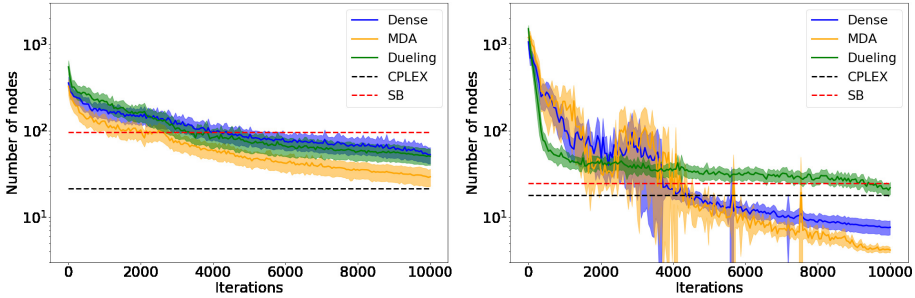
Fig. 2. Cross-validated performance on test instances (averaged number of nodes in log scale) for $\mathcal{P}_1$ (left) and $\mathcal{P}_2$ (right) through iterations of Algorithm 1. Gaussian confidence intervals are shown around the means.

As exhibited in Fig. 2, our method is able to learn an efficient strategy from scratch. As expected, the MDA agent is more flexible than its additive counterpart (Dueling) inspired from the Dueling architecture of [16] and the fully-connected agent (Dense). It outperforms systematically the Strong Branching policy, and finds comparable or better strategies than CPLEX, depending on the problem. Results on training data are not displayed for the sake of conciseness, but it is worth mentioning that our agents do not overfit and are able to generalise well. In addition, the computation time is negligible compared with full Strong Branching as an action comes only at the price of a forward pass in our neural network.

Despite these good performances, some limits have to be pointed out at this stage. First, our framework requires DFS as the node selection strategy, which can be far from optimal for certain problems. Note that using another strategy may be complicated to handle due to more complex dependencies, but may also turn out to be effective as targetting small subtrees makes sense in general. Second, we only showed promising results on easy problems. With more difficult problems, the training becomes computationally prohibitive as a randomly initialised agent produces exponential trees. To tackle these limitations, we encompass different solutions such as fine-tuning the features and network architecture or using supervision to decrease the size of the generated trees during the first episodes. To reduce the cost of exploration, one could apply the same methodology with a set of branching heuristics as action set, similarly to what is proposed in [18].

6   Conclusion

In this paper, we presented a novel Reinforcement Learning framework designed to learn from scratch the branching strategy in a B&B algorithm. In addition to the specific metrics used in our FMSTS method, we introduced a new neural network architecture designed to tackle the multiplicative nature of the value function. Besides, we adapted some known RL techniques to the B&B setting.

We ran experiments on real-world problems to validate our method and showed better or comparable performances with existing strategies.

It is worthwhile to highlight that our method is generic enough to be applied to other metrics than the tree size, e.g. the number of simplex iterations or even the computation time. If one is not interested in proving optimality, many other value functions may be encompassed. Furthermore, it may be interesting to enlarge the scope of the method, especially to include Branch and Cut algorithms as they usually are more efficient.

References

1. Wolsey, L.A.: Integer programming. Wiley, Hoboken (1998)
2. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. arXiv preprint arXiv:1811.06128 (2018)
3. Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W., Vance, P.H.: Branch-and-price: column generation for solving huge integer programs. Oper. Res. **46**(3), 316–329 (1998)
4. Mitchell, J.E.: Branch-and-cut algorithms for combinatorial optimization problems. In: Handbook of Applied Optimization, vol. 1, pp. 65–77 (2002)
5. Rachelson, E., Abbes, A.B., Diemer, S.: Combining mixed integer programming and supervised learning for fast re-planning. In: 2010 22nd IEEE International Conference on Tools with Artificial Intelligence, vol. 2, pp. 63–70. IEEE (2010)
6. Khalil, E.B., Le Bodic, P., Song, L., Nemhauser, G., Dilkina, B.: Learning to branch in mixed integer programming. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
7. Balcan, M.-F., Dick, T., Sandholm, T., Vitercik, E.: Learning to branch. arXiv preprint arXiv:1803.10150 (2018)
8. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. arXiv preprint arXiv:1906.01629 (2019)
9. Tang, Y., Agrawal, S., Faenza, Y.: Reinforcement learning for integer programming: learning to cut. arXiv preprint arXiv:1906.04859 (2019)
10. He, H., Daume III, H., Eisner, J.M.: Learning to search in branch and bound algorithms. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 27, pp. 3293–3301. Curran Associates Inc., New York (2014)
11. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (2018)
12. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
13. Minsky, M.: Steps toward artificial intelligence. Proc. IRE **49**(1), 8–30 (1961)
14. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. arXiv preprint arXiv:1511.05952 (2015)
15. Pearson, K.: LIII. On lines and planes of closest fit to systems of points in space. London Edinburgh Dublin Philos. Mag. J. Sci. **2**(11), 559–572 (1901)
16. Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., De Freitas, N.: Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581 (2015)
17. Manual, C.U.: IBM ILOG CPLEX Optimization Studio (1987)
18. Di Liberto, G., Kadioglu, S., Leo, K., Malitsky, Y.: Dash: dynamic approach for switching heuristics. Eur. J. Oper. Res. **248**(3), 943–953 (2016)