Using Graph Convolution Neural Networks for Solving Mixed integer Linear Programs

Nguyen Thi My Binh[1], Dao Hoang Long[2], Nguyen Van Thien[1], Pham Hong Thai[1]
[1] Hanoi University of Industry
[2] Hanoi University of Science and Technology

Tác giả liên hệ: Nguyen Thi My Binh, Email: binhntm@haui.edu.vn
Ngày nhận bài: 09/08/2022, ngày sửa chữa: 17/11/2022, ngày duyệt đăng: 25/11/2022
Định danh DOI: 10.32913/mic-ict-research vn.v2022.n2.1130

Tóm tắt: This paper focuses on an exact algorithm for solving NP-hard combinatorial optimization problems which frequently requires significant specialized knowledge and trial and error, especially, Mixed Integer Linear Programs (MILP). This challenging, tedious process can be automated by learning the algorithms instead. In many practical applications, the same optimization problem is tackled again and again on regular basis, maintaining the same problem structure but varying in the data. This offers an opportunity for learning algorithms that employ the structure of such recurrent problems. With the motivation, we present a network model for learning branch-and-bound variable selection policies based on reinforcement learning, which leverages the natural variable constraint bipartite graph representation of MILPs. The model is trained going through mimic learning from the strong branching rule. Experimental results claim that using graph convolutional neural networks for solving MILPs can improve for designing branching rules to solve large problems, and outperforms prior models.

Từ khóa: *Graph convolution neural network, mixed integer linear programs, branch and bound.*

Title: **Sử dụng mạng tích chập đồ thị để giải các bài toán quy hoạch nguyên hỗn hợp**

Abstract: Nghiên cứu này tập trung vào giải quyết bài toán quy hoạch nguyên hỗn hợp (MILP), bài toán tối ưu hóa tổ hợp thuộc lớp NP-Hard bằng giải thuật chính xác. Để giải quyết lớp bài toán này, chúng ta cần có những kỹ thuật đặc biệt, thường đòi hỏi chuyên môn cao về kiến thức và thử nghiệm. Quy trình giải các bài toán này có thể được tự động hóa bằng cách sử dụng các thuật toán có khả năng tự học. Trong nhiều ứng dụng thực tế, các bài toán tối ưu hóa được giải quyết giống nhau trên cùng một nền tảng và cấu trúc nhưng thay đổi trong dữ liệu. Điều này cung cấp một cơ hội để học các thuật toán sử dụng cấu trúc của các vấn đề lặp đi lặp lại như vậy. Do đó, chúng tôi trình bày một mô hình mạng cho việc học chính sách phân nhánh cho bài toán nhánh cận dựa trên học tăng cường, tận dụng tích chất có thể biểu diễn dưới dạng đồ thị hai chiều giữa biến và ràng buộc của MILP. Mô hình được huấn luyện thông qua bắt chước học từ quy tắc phân nhánh mạnh. Kết quả thực nghiệm cho thấy sử dụng mạng tích chập đồ thị để giải quyết MILP có thể cải thiện cho vấn đề thiết kế các luật phân nhánh để giải các bài toán với kích thước dữ liệu lớn, và vượt trội so với các mô hình trước đó.

Keywords: *Mạng tích chập đồ thị, quy hoạch nguyên hỗn hợp, nhánh cận.*

I. INTRODUCTION

Combinatorial optimization (CO) problems occurs in various of practical applications, such as scheduling, routing, assignment,cutting and packing, network design, protein alignment, and many other important domains like economic, industrial and scientific. CO problems can be classified into twofolds as discrete and continuous CO problems. Most discrete CO problems (DCOPs) are NP-Hard problems which could not be solved in polynomial time [1]. The available techniques for DCOPs can be categorized into two main: exact and heuristic methods [2].

Mixed-Integer Linear Programs (MILPs) are typical of combinatorial optimization problems. Most CO problems can be formulated as MILPs which receive the attention of many operations researchers. This interest is due in part to the practical importance of problems, but also to its intrinsic difficulty [2]. Several exact algorithms such as branch-and-bound (B&B), dynamic programming, Lagrangian relaxation based methods etc., are guaranteed to find an optimal solution and to prove its optimality for

every instance of MILPs. However, the running execution time often increases significantly with the instance size, and often only small or moderately-sized instances can be tackled to provable optimality. In addition, the ability to execute efficient algorithms is extremely important for solving real-world large-scale combinatorial optimization challenges encountered in many established and emerging heterogeneous areas.

Recently, machine learning (ML) has seen a surge in the development and especially in the deep learning and deep reinforcement learning areas [3]. This has led to significant performance improvements on many tasks within diverse fields. ML accomplishes the imperative require to efficiently solve practical combinatorial optimization problems regarding execution time and quality of the solutions. In many practical application areas we have to access to data of unprecedented scale and accuracy about the system/ process we want to model. Furthermore, ML provides techniques to exploit available data and extract value and useful information, which can be employed for modeling and solving combinatorial problems. This a major driving force for devising innovative solutions to combinatorial challenges. Furthermore, the potential of ML for addressing CO problems is examined by academy community. Recently, novel and advanced techniques for solving CO problems have been strongly developed in ML area [4]. Noticeably, the inherent structure of the CO problems in numerous fields or the data itself is that of a graph [5]. In this paper focuses on applying graph convolution neural network (GCNN) for solving CO problems in generally, mixed-integer linear problems in particularly [6]. This paper focuses on tackling MILPs by exact algorithm known as Branch and Bound (B& B). The idea of B& B algorithm is that it recursively divides the solution space into a search tree, and relaxes bounds along the way to prune subtrees which probably cannot obtain an optimal solution. B& B is a type of tree search algorithms which is widely used tool for solving CO problems. However, in many circumstances it is usually to repeatedly resolver similar combinatorial optimization problems, e.g., scheduling, and day-to-day production planning problems [7], which may strongly differ from the set of instances on which B&B algorithms are typically evaluated. It is necessary to use statistical learning for fine-tune B&B algorithms automatically for a specific class of problems. However, this work exists two challenges. The first one, it is not obvious how to encode the state of a MILP in B&B decision process [8], especially because both search trees and integer linear programs can have a variable structure and size. The second one, it is not show how to formulate a model architecture that derives from rules which can be generalized, at least to similar circumstances. To overcome these shortages, this paper represents how to use graph convolution neural networks for solving MILP problems. More precisely, variable collection in branching problem is considered two aspects as follows:

- How to encode the branching policies into a graph convolution neural network (GCNN), which help with exploit the natural bipartite graph to represent MILP problems.
- Strong branching decisions are approximated by using behavioral cloning with a cross-entropy loss.

The rest of the paper is organized as follows: Related works are presented in Section 2. Background and formulate the branching problem in Section 3. Section 4 presents methodology for solving the branching problem. Experimental results are discussed in Section 5. Finally, Section 6 presents conclusions

II. RELATED WORKS

This section briefly review literature of works that use of machine learning techniques in the context of branching. The authors in [9] focused on variable selection. Their goal is to search a variable selection strategy that intimates the behavior of the classical branching strategy known as strong branching while running faster than strong branching. Alvarez et. al. [8] studied a regression problem and learn directly the strong branching scores of indicates. They then proposed the ExtraTrees which are very robust with respect to the choice of their parameters. Furthermore, in their work, the feature vectors in the training set describe nodes from multiple MILP instances. In [5], the authors was the first proposed a GCNN model for learning greedy heuristics on NP-hard combinatorial optimization problems which are defined on graphs. Selsam et al. in [10], also studied GCNN model, NeuroSAT which can be interpreted a message passing neural network that learns to tackle SAT problem after being trained as classifier to predict satisfiability.

Many works exploited data-driven variable selection from a purely experiment. Karzan et al.[11] divided techniques for picking problem-specific branching rules based upon a partial B&B tree. These branching rules will choose variables that will lead to fast discovering. For CSP tree search, Xia et. al. [12] implemented existing multi-armed bandit algorithms to learning variable selection policies during tree search. In [13], Balafrej et al. applied a bandit approach to choose different levels of propagation during search.

Other works focus on using ML to improve variable selection in B&B, without directly learning a branching policy. The authors in [14] studied how to use ML to determine

an optimal weighting of any set of partitioning procedures for the circumstance distribution at hand using samples. In [15], Liang et al. investigated a variable selection policy for solving SAT problem. They then modeled the variable selection optimization problem as an online multi-armed bandit, a special-case of reinforcement learning, to learn branching variables such that the learning rate of the solver is maximized.

Several works investigate using machine learning techniques in other aspects of B&B beyond variable selection [5, 16, 17]. Authors in [16] focused on learning a node selection heuristic by mimic learning of the expert procedure that widens the node whose feasible set including the optimal solution. In [18], Song et al. studied for node selection and pruning heuristics by mimic learning of shortest paths to obtain good feasible solutions. Khalil et al. [5] presented a theoretical framework for analyzing this decision-making process in a simplified setting, proposed a ML approach for modeling heuristic success likelihood, and designed practical rules that leverage the ML models to dynamically decide whether to run a heuristic at each node of the search tree. Those methods could in principle be combined together to further improve performance for solving combinatorial optimization problems. In addition, many works have divided machine learning approaches to fine-tune exact optimization algorithms.

III. BACKGROUND

1. Mixed integer linear program

We consider a mixed integer linear program which is defined as optimization problem as follows:

$$\arg\min_{x}\{c^T x | Ax \leq b, \mathrm{l} \leq \mathrm{x} \leq \mathrm{u}, \mathrm{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}\}, \quad (1)$$

where $c \in \mathbb{Z}^n$ denotes the objective coefficient vector, $A \in \mathbb{R}^{m \times n}$ is the constraint coefficient matrix, $b \in \mathbb{R}^n$ denotes the constraint right-hand-side vector, $l; u \in \mathbb{R}^n$ are the lower and upper variable respectively, $p \leq n$ is the number of integer variables. To solve MILP in (1), the B&B algorithm is applied to integer programming by relaxing the integrality constraint. A continuous linear program (LP) is obtained whose solution by providing a lower bound to MILP in (1). In addition, LP can be tackled efficiently by the simplex algorithm as well. If a solution to the LP relaxation respects the original integrality constraint, then it is also a solution to the problem in (1). If not, then one may decompose the LP relaxation into two sub-problems, by splitting the feasible region according to a variable that does not respect integrality in the current LP solution x^*,

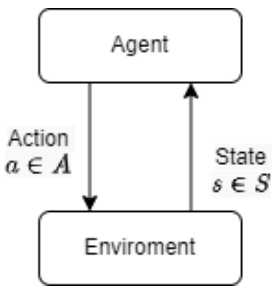
$$x_i \leq \lfloor x_i^* \rfloor \vee x_i \geq \lceil x_i^* \rceil \exists \mathrm{i} \leq \mathrm{p} | \mathrm{x}_i^* \notin \mathbb{Z} \quad (2)$$

where $\lfloor . \rfloor$ and $\lceil . \rceil$ respectively denote the floor and ceil functions. In practice, the two sub-problems will only differ from the parent LP in the variable bounds for x_i, which get updated to $u_i = \lfloor x_i^* \rfloor$ in the left child and $l_i = \lceil x_i^* \rceil$ in the right child.

Applying the B & B algorithm [14], in its simplest formulation, this binary decomposition is performed repeatedly, giving rise to a search tree. The best solution of LP is in the leaf nodes of the tree, which provides a lower bound to the original MILP, whereas the best integral LP solution (if any) provides an upper bound. The stop conditions of B&B algorithm can be obtained whenever both the upper and lower bounds are equal or when the feasible regions do not decompose anymore, as of result providing a certificate of optimality or infeasibility, respectively.

2. The branching problem formulation as a Markov decision process

The B & B algorithm for solving MILP makes the sequential decisions which can be become apart of a Markov decision process [16]. Let the agent be considered the brancher, the solver be the environment. The decision of the solver at the t^{th} and in a state s_t consist of the B & B tree with all past branching decisions. The best integer solution was fathomed so far, the LP solution of each node, the currently focused leaf node, as well as any other solver statistics. The brancher then picks to a variable at among all fractional variables $\mathcal{A}(s_t) \subseteq \{1, 2, ..p\}$ at the currently focused node, according to a policy $\pi(a_t|s_t)$. The solver sequentially extends the B&B tree, tackles the two child LP relaxations, executes any internal heuristic, cuts off the tree if warranted, and finally chooses the next leaf node to split. The algorithm is then in a new state s_{t+1}, and the brancher is called again to take the next branching decision, this progress still executes until the instance is addressed, that is,stop condition of B & B algorithm is no leaf node left for branching. Figure 1 depicts the Markov decision process.



Hình 1. Illustration of the Markov decision process

Episodic is all states that come in from an initial state to terminal state. B & B algorithm is a Markov decision pro-

cess and episode as well. Each episode includes of amount of solving a MILP instanse. Beginning states correspond to an instance initial sampled among a group of interest, while ending states mark the finish of the optimization process. The probability of a trajectory $s = (s_0, s_1, ...s_{\mathcal{T}}) \in \mathcal{T}$ then depends on both the branching policy π and the remaining components of the solver,

$$p_\pi((T)) = p(s_0) \prod_{t=0}^{\mathcal{T}} \sum_{a \in \mathcal{A}(s_t)} \pi(a|s_t) p(s_{t+1}|s_t, a) \quad (3)$$

Reinforcement learning with a carefully designed reward function is a natural approach to search good branching policies.

IV. METHODOLOGY

This section presents methodology for solving MILP by B & B. In details, we use a mimic learning and a GCNN model for B & B algorithm. B & B is type of tree search algorithms. These algorithms recursively partition the search space to find an optimal solution. To keep the tree small, it is so important to carefully decide, when expanding a tree node, which variable to branch on at that node to partition the remaining space. Therefore, variable selection problem (VSP) plays an important role in solving MILP by B & B. VSP can be formulated as a Markov decision process, a natural way of training a policy would be reinforcement learning [19].

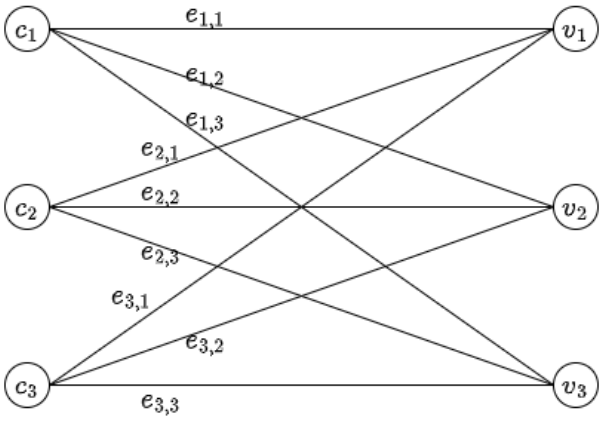
Mimic learning: Model training intimates the strong branching rule [6]. First, the expert on a collection of training instances of interest is run, a dataset of expert state-action pairs is then recorded $Q = \{(s_i, a_i^*)\}$. Second, learn policy should be minimized the cross-entropy loss

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{(s,a_*)} \log_{\pi_0}(a_*|s) \quad (4)$$

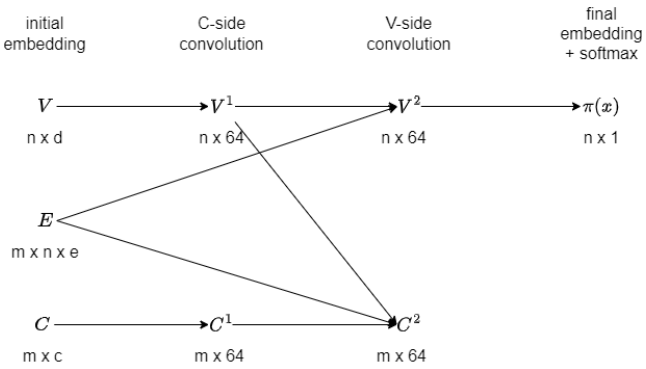
Encoding: the state s_t of the B&B process at time t as a bipartite graph with node and edge features $(G; C; E; V)$ is encoded. Figure 2 demonstrates the graph are nodes corresponding to the constraints in the MILP, in which one per row in the current node's LP relaxation is their feature matrix $C \in \mathbb{R}^{m \times c}$.

Figure 3 illustrates nodes with respect to the variables in the MILP, in which one per LP column, with $V \in \mathbb{R}^{n \times d}$ their feature matrix. An edge (j, j) belongs to E if it links a constraint node i and a variable node j, mean that $A_{ij} \neq 0$ and $E \in \mathbb{R}^{m \times n \times e}$ shows features of edge.

Parametertrization : The variable selection policy $\pi_0(a, s_t)$ is paramerized as a graph convolution neural network (GCNN). To overcome this issue and stabilize the learning procedure, we adopt a simple affine transformation



Hình 2. Illustration of the bipartite state $s_t = (G; C; E; V)$ with $n = 3$ variables and $m = 2$ constraints



Hình 3. Demonstration of the architecture of bipartite GCNN for parametrizing the policy $\pi_\theta(a|s_t)$

$x \leftarrow (x - \beta)/\sigma$, which is *pnorm layer* (The β and σ parameters are initialized with respectively the empirical mean and standard deviation of x on the training dataset)

Model: The base of our GCNN model includes: *pnorm layer*, 2-layer perceptrons with relu activation functions, and bipartite graph convolution

V. EXPERIMENT

1. System and data settings

a) System

We train and evaluate in Ubuntu 20.04.4 (LTS) operating system. This server have configuration:

- Memory: 400GB
- CPU: Intel(R) Xeon(R) Gold 5220 CPU @ 2.20GHz

b) Data settings

The GCNN approach for MILP is evaluated on an NP-hard benchmark as set covering instances which are generated in [20] with 1,000 columns. For more details, each column represents for a variable while each row

represents for a constraint. Training and testing processes are employed on instances with 500 rows. Furthermore, the difficulty of the training set is based on the number of rows, so we evaluate GCNN model on instances with 500 (Easy), 1,000 (Medium) and 2,000 (Hard) rows.

2. Experimental results

GCNN is compared against two prior machine learning branchers as (LMART) [21] and (TREES) [8]. The TREES model uses variable-smart features from bipartite state, obtained by concatenating the variable's node features with edge and constraint node features statistics over its neighborhood. The LMART model uses re-implemented within SCIP. Table I, II and III contain the results about time consumption, a number of wins for each instances, and a

Bảng I
THE EVALUATION OF POLICY ON EASY INSTANCES REGARDING RUNNING TIME (TIME), A NUMBER OF WINS (WINS), AND A NUMBER OF BRANCHING NODES (NODES)

Model	Time	Wins	Nodes
TREES	9.28	0/100	187
LMART	7.19	14/100	167
GCNN	**6.59**	**85**/100	**134**

In Table I, the results show a great challenge which is put on the TREES and LMART models by easy samples. However, GCNN can handle it easily and it outperforms the other algorithms at every index. It takes a high number of wins which is 85 out of 100 instances and also costs the least time and number of branching nodes.

Bảng II
THE EVALUATION OF POLICY ON MEDIUM INSTANCES REGARDING RUNNING TIME (TIME), A NUMBER OF WINS (WINS), AND A NUMBER OF BRANCHING NODES (NODES)

Model	Time	Wins	Nodes
TREES	92.47	0/100	2187
LMART	59.98	0/100	1925
GCNN	**42.48**	**100**/100	**1450**

In Table II, GCNN still performs incredibly. The results show that TREES and LMARTS models totally give up before medium instances. However, GCNN amazingly solves each one of them with lower time and less branching nodes than the rest algorithms. Another notable result is the time, wins and nodes of GCNN against medium samples is higher than the easy one, it makes a hypothesis that the higher branching nodes GCNN creates, the more chances to solve the problem.

Bảng III
THE EVALUATION OF POLICY ON HARD INSTANCES REGARDING RUNNING TIME (TIME), A NUMBER OF WINS (WINS), AND A NUMBER OF BRANCHING NODES (NODES)

Model	Time	Wins	Nodes
TREES	2869.21	0/35	59013
LMART	2165.96	0/54	45319
GCNN	**1489.91**	**66**/70	**29981**

In table III, the results are similar to table I and table II. However, GCNN does not completely beat the hard instances like it has done to the medium ones. It proves that GCNN still has a limit, which means an instances with numerous rows can realize make trouble for GCNN. Therefore, GCNN shows the best result with instances having from 1000 to 2000 rows.

VI. CONCLUSIONS

This paper presents an exact method, brand and bound algorithm for solving mixed integer linear programs as a Markov decision process. By transform the state of branching process into as a bipartite graph, which reduces the required feature engineering by leveraging the variable constraint structure of mixed integer linear programs naturally. We implement experiments on an NP-hard, especially, a set covering problem, by adopting a mimic learning scheme. We then analyze, evaluate a new approach through extensive experiments. The results show that the policies learned of GCNN do better than prior machine learning approaches, even could outperform the standard branching strategy of SCIP. Furthermore, the learned policies of GCNN are able to generalize to instance sizes larger than seen during training. This result is due to collecting strong branching decisions, however, training, can be computationally too expensive on large instances. In short, the GCNN model is better architecture than previous model for the task branching in mixed integer linear program.

ACKNOWLEDGEMENT

This research is funded by Hanoi University of Industry under grant number 25- 2022- RD/HD /- DHCN for Nguyen Thi My Binh.

TÀI LIỆU THAM KHẢO

[1] B. H. Korte, J. Vygen, B. Korte, and J. Vygen, *Combinatorial optimization*, vol. 1. Springer, 2011.
[2] S. Martello, M. Minoux, C. Ribeiro, and G. Laporte, *Surveys in combinatorial optimization*. Elsevier, 2011.
[3] N. Vesselinova, R. Steinert, D. F. Perez-Ramirez, and M. Boman, "Learning combinatorial optimization on graphs: A survey with applications to networking," *IEEE Access*, vol. 8, pp. 120388–120416, 2020.
[4] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour

d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.
[5] E. B. Khalil, B. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao, "Learning to run heuristics in tree search.," in *Ijcai*, pp. 659–666, 2017.
[6] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[7] Y. Pochet and L. A. Wolsey, *Production planning by mixed integer programming*, vol. 149. Springer, 2006.
[8] A. M. Alvarez, Q. Louveaux, and L. Wehenkel, "A machine learning-based approximation of strong branching," *INFORMS Journal on Computing*, vol. 29, no. 1, pp. 185–195, 2017.
[9] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina, "Learning to branch in mixed integer programming," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.
[10] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, "Learning a sat solver from single-bit supervision," *arXiv preprint arXiv:1802.03685*, 2018.
[11] F. K. Karzan, G. L. Nemhauser, and M. W. Savelsbergh, "Information-based branching schemes for binary linear mixed integer problems," *Mathematical Programming Computation*, vol. 1, no. 4, pp. 249–293, 2009.
[12] W. Xia and R. Yap, "Learning robust search strategies using a bandit-based approach," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.
[13] A. Balafrej, C. Bessiere, and A. Paparrizou, "Multi-armed bandits for adaptive constraint propagation," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
[14] M.-F. Balcan, T. Dick, T. Sandholm, and E. Vitercik, "Learning to branch," in *International conference on machine learning*, pp. 344–353, PMLR, 2018.
[15] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning rate based branching heuristic for sat solvers," in *International Conference on Theory and Applications of Satisfiability Testing*, pp. 123–140, Springer, 2016.
[16] H. He, H. Daume III, and J. M. Eisner, "Learning to search in branch and bound algorithms," *Advances in neural information processing systems*, vol. 27, 2014.
[17] A. Sabharwal, H. Samulowitz, and C. Reddy, "Guiding combinatorial optimization with uct," in *International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming*, pp. 356–361, Springer, 2012.
[18] J. Song, R. Lanka, A. Zhao, A. Bhatnagar, Y. Yue, and M. Ono, "Learning to search via retrospective imitation," *arXiv preprint arXiv:1804.00846*, 2018.
[19] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.
[20] E. Balas and A. Ho, *Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study*, pp. 37–60. Berlin, Heidelberg: Springer Berlin Heidelberg, 1980.
[21] C. Hansknecht, I. Joormann, and S. Stiller, "Cuts, primal heuristics, and learning to branch for the time-dependent traveling salesman problem," *arXiv preprint arXiv:1805.01415*, 2018.

AUTHOR'S SUMMARY

Nguyen Thi My Binh is a lecturer of the Faculty of Information Technology, Hanoi University of Industry, Vietnam. She received the M.Sc. degree in Information Technology in 2005, and the PhD. degree in Computer Science at Hanoi University of Science and Technology, Vietnam. Her current research interests include optimization, computational intelligence, Artificial Intelligence.
Email: binhntm@haui.edu.vn or binhntm@fit-haui.edu.vn

Dao Hoang Long graduated Computer Science from Hanoi University of Science and Technology. His research interests include application of heuristic techniques in solving NP-hard problems and applying real-time deep learning in software engineering.
Email: long.dh2000.bachkhoa@gmail.com

Nguyen Van Thien graduated at the Hanoi University of Science and Technology in 1998. He received the M.Sc. degree in Mathematics and Informatics at the Hanoi National University of Education in 2000. He gained the PhD. degree at the Institute of Information Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam. His current research interests include intelligent information system, database, and data mining.
Email: nguyenthien@haui.edu.vn

Pham Hong Thai Pham Hong Thai received Bachelor of Information Technology at the Hanoi University of Industry. He is a Master's student in Information System at Hanoi University of Industry. He is passionate about artificial intelligence.
Email: thaiph99@gmail.com