2025 5th International Conference on Artificial Intelligence and Industrial Technology Applications (AIITA)

# Learning to Configure Hyperparameters in Solving Unit Commitment Problems with Mixed Integer Linear Programming Solvers

Zhoujun Ma1st
State Grid Jiangsu Electric Power Co., Ltd
Nanjing, China

Jinbo Liu2nd
State Grid Corporation of China
Beijing, China

Yishen Wang3rd
Artificial Intelligence Institute, China Electric Power Research Institute
Beijing, China
wangyishen07@163.com

Renjie Wei4th
Artificial Intelligence Institute, China Electric Power Research Institute
Beijing, China
wrj245478404@hotmail.com

Fei Xue5th
Suqian Power Supply Branch,
State Grid Jiangsu Electric Power Co., Ltd
Suqian, China

Zhaoye Pan6th
Alibaba Group
Hangzhou, China
panzhaoye.pzy@alibaba-inc.com

Yuanxin Yu7th
Alibaba Group
Hangzhou, China
yuyuanxin.yyx@alibaba-inc.com

Mengchang Wang8th, *
Alibaba Group
Hangzhou, China
*mengchang.wmc@alibaba-inc.com

***Abstract*—Unit commitment problems can be solved more efficiently with mixed integer linear programming solvers when more preferred hyperparameters are configured. We propose a learning approach to configure hyperparameters automatically for unit commitment problems. Our method employs a bipartite graph to represent the underlying structure of MILP and leverages a Graph Convolutional Network to adjust solver parameters on an instance-by-instance basis. The effectiveness of this approach is empirically validated through a series of numerical experiments, which demonstrate significant improvements in solver performance, resulting in noticeable computational time savings. These findings highlight the practicality and potential of our strategy in real-world applications of unit commitment, suggesting a promising new direction for optimization solver configuration.**

***Keywords-Unit Commitment, Mixed Integer Programming, Solver Hyperparameters, Graph Convolutional Network***

## I. INTRODUCTION

Unit commitment problems (UCs) are to determine the on/off status and the generation power of each unit attached to a power grid during the next day, which are usually solved with mixed integer linear programming (MILP) solvers (e.g., CPLEX, Gurobi, MindOpt, etc.).

These solvers are often equipped with branch-and-cut (B&C) algorithms and plenty of heuristics, where configurable hyperparameters are built inside to control the algorithmic behavior [1]. The default hyperparameter configurations are determined by solver developers for the needs of solving general MILP models, which could not be the best for specific problem types. By carefully tuning hyperparameters against specific MILP instances, the solving speed is reported to be accelerated by more than 100 times [2]. It is reasonable to hypothesize that hyperparameter tuning can also contribute to accelerate UC solving.

However, due to the ever-changing demands and dynamic grid topology, hyperparameter configurations tuned for the UC of one day may not necessarily be suitable for UCs on the other days. The generalization capability of hyperparameter configurations deteriorates further with the increasing integration of fluctuating renewable energy sources into power grids. Experiments indicate that suboptimal configurations can significantly degrade solver performance, potentially causing dramatic slowdowns [2].

There are usually two ideas for dealing with those situations: (1) tuning hyperparameters against a large number of UCs to improve the generalization capability of the obtained configurations, and (2) re-using the configuration of a similar UC instance that has ever been well tuned [1]. The former way will in fact ignore specific characteristics of each UC instance, leading to robust but mediocre hyperparameter configurations that may be too conservative to achieve any remarkable acceleration. The latter way is more reasonable when the changes in demands and topology possess certain periodicity or seasonality, which happens to be the case in practice [3]

In this paper, we propose a learning approach to automatically suggest hyperparameter configurations based on a measurement of the similarity between a UC instance and those well-tuned ones in the past. This approach fully considers the specific characteristics of each UC instance and its similarity with other UC instances, and can customize acceleration strategies for UC instances with different demands and topology. A concise review of relative work is presented in Section II. In Section III, the learning approach is described in detail, which is followed by numerical experiments in Section IV, demonstrating the effectiveness and the capability of generalization of the proposed approach. Finally, we come to a conclusion in Section V.

## II. Relative Work

Hyperparameter configuration has attracted extensive studies as reviewed in [4-6]. One of the most popular tuning algorithms is Lindauer et al.'s SMAC3 [7]. Recently, Himmich et al. propose a multi-phase iterative local search method, starting from a small parameter pool that is dynamically enriched, instead of exploring the original configuration space [8].

Applying MindOpt Tuner [2], a distributed hyperparameter tuning tool, on an open source solver, Cbc, more than 100 times acceleration is observed on some MIPLIB2017 instances, against the default hyperparameter configurations, while the total tuning time is significantly reduced compared with SMAC3. With hyperparameter tuning, the performance of CPLEX is dramatically enhanced over the 240 MILP instances of MIPLIB2017 [9].

Considering that the hyperparameter tuning processes could be time-consuming and sometimes impossible to do against an incoming MILP instance due to the limited time budget, machine learning approaches are studied in order to configure solvers in a real-time manner. Among those studies, Malitsky et al. propose an instance-aware configuration method, ISAC, which clusters problem instances based on their feature vectors consisting of the problem sizes, coefficients in the cost functions, constraint matrices, right-hand-sides, etc [10]. Xu et al. additionally include information from short solver runs in feature vectors before predicting the final configurations [11]. Hosny and Reda adopt Graph Convolutional Networks (GCNs) and Deep Metric Learning methods to learn the features and MILP similarities [1].

Despite these advancements, the existing literature has predominantly overlooked the nuanced demands of configuring hyperparameters specifically for unit commitment problems. Given the unique characteristics inherent to these challenges, there is a pressing need for a targeted approach. As depicted in Fig. 1, several experiments are conducted with different parameter configurations, and the results indicate that the worst-performing configuration takes approximately twice as long as the best-performing configuration when applied to a rudimentary unit commitment problem. This gap underlines the necessity for a specialize framework—Learning to Configure Hyperparameters in Solving Unit Commitment Problems with Mixed Integer Linear Programming Solvers. Such an endeavor aims to tailor hyperparameter tuning processes to the specific requirements of unit commitment issues, ensuring that optimization is both precise and attuned to the intricacies of these complex problem sets.

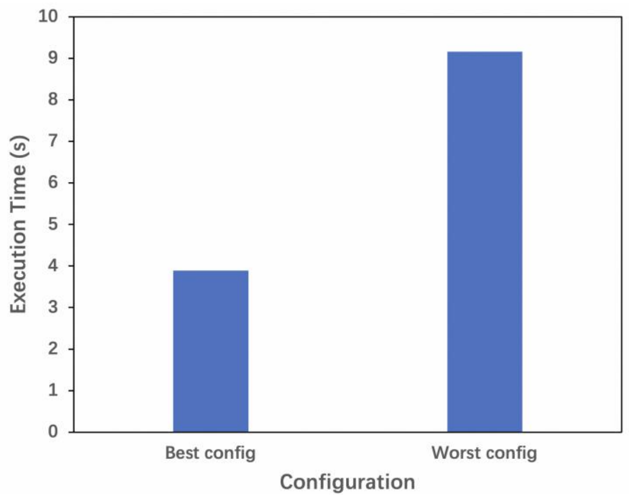

Figure 1. Importance of Correctly Setting Config for UC

## III. A Learning Approach

In this section, we develop a learning approach for configuring hyperparameters in solving unit commitment problems with mixed integer linear programming solvers, which consists of using a bipartite graph to represent the mixed integer programming structure and construct a Graph Convolutional Network for instance-wise tuning solver's parameters.

### A. MILP Representation

Considering the typical unit commitment problem, which entails the determination of the optimal scheduling and dispatch of power-generating units over a forthcoming time horizon, typically spanning from several hours to several days, has the following mathematic representation as Equation 1.

$$\begin{aligned}
min \quad & \sum_{t=1}^{T}\sum_{g=1}^{G}\left(x_{gt}C_g^{fixed} + w_{gt}C_g^{start} + p_{gt}C_g^{var}\right) \\
s.t. \quad & \sum_{k=t-L_g+1}^{t} w_{gk} \le x_{gt}, \forall g, \forall t = L_g - 1, \dots, T-1 \\
& \sum_{k=g-l_g+1}^{T} w_{gt} \le 1 - x_{g,t-l_g+1}, \forall g, \forall t = l_g - 1, \dots, T-1 \\
& w_{gt} \ge x_{gt} - x_{g,t-1}, \forall g, \forall t = 1, \dots, T-1 \\
& \sum_{g=1}^{G} p_{gt} \ge D_t, \forall t \\
& P_g^{min} x_{gt} \le p_{gt}, \forall g, t \\
& p_{gt} \le P_g^{max} x_{gt}, \forall g, t \\
& x_{gt} \in \{0,1\}, \forall g, t
\end{aligned} \tag{1}$$

In the above formulation, $T$ is the number of time horizons, $G$ is the total number of generation units, and $D_t$ is the power demand at timescale $t$. For each unit, $P_g^{max}$ and $P_g^{min}$ are the maximum and minimum output power of the corresponding unit $g$, $L_g$ and $l_g$ are the minimum up-time and down-time of the corresponding unit $g$. In the formulation of the objective function, it is commonly assumed that production costs exhibit a linear relationship with the generated output. We denote $C_g^{fixed}$ as the fixed cost incurred for maintaining a generating unit in an operational state for a single time step, irrespective of its power output; $C_g^{start}$ represents the cost associated with initiating the operation of the unit, also known as the start-up cost; and $C_g^{var}$ reflects the variable cost per megawatt for the generation of electricity by the unit under consideration. Considering the decision variable, define $x_{gt}$ as a binary variable that assumes a value of one if and only if the unit $g$ in question is operational at timescale $t$, whereas $w_{gt}$ denotes a binary variable that equals one exclusively when the unit transitions from a non-operational state at time $t-1$ to an operational state at time $t$. Additionally, let $p_{gt}$ be a continuous variable representing the quantity of power generated by the unit.

The application of bipartite graphs to MIP problems facilitates a clear and concise depiction of the interplay between variables and constraints. In the context of MIP, one subset of vertices represents the decision variables, which may include both continuous and discrete (integer or binary) variables, while the other subset embodies the constraints of the problem. Edges are drawn between a variable and a constraint if the variable figures in the linear expression constituting the constraint.

The utility of employing a bipartite graph to represent an MIP is multifold. Primarily, it elucidates the structure of the MIP's constraint matrix by visually identifying which variables are involved in each constraint, thus aiding in comprehending the overall sparsity or density of the system. Furthermore, bipartite graphs can be pivotal in revealing patterns of variable-constraint connectivity, such as identifying substructures within the problem that may be exploited for computational efficiency through decomposition techniques.

In summary, the adoption of bipartite graphs in representing mixed integer programming problems offers a compelling approach to visualizing and analyzing the complex relationships between variables and constraints, ultimately enabling a deeper understanding and more efficient resolution of these challenging optimization problems.

In our approach, we utilize the bipartite graph to represent the unit commitment problem. Specifically, we define a weighted bipartite graph $\mathcal{G} = (\mathcal{V} \cup \mathcal{W}, \mathcal{E})$ in the following manner, as elaborated in [12].

1) Each vertex in constraint vertex set is associated with the constraints in unit commitment problem, and the attribute of each vertex is encapsulated by its bias term.

2) Each vertex in variable vertex set is associated with the variables in the unit commitment problem. The characteristic feature of vertex is represented by a vector, which encompasses details regarding the objective coefficient, the nature of the variable and physical information.

3) Each edge in edge set connects a constraint vertex and a variable vertex. The characteristic feature of edge is represented by the problem's coefficient.

## *B. GNN*

Graph Neural Networks (GNNs) present an advanced and potent framework for dissecting and interpreting the intricate relational structures frequently encountered in the domain of Operations Research. By capitalizing on their capacity to capture and model the nuanced interdependencies within data, GNNs have emerged as a pivotal instrument for tackling the multifaceted challenges inherent in this field. Reference [13] introduced using Graph Neural Networks (GNNs) for combinatorial optimization. This approach employs the structure2vec GNN to model the graph structure of the current solution and computes Q-values for each remaining optional node. A new node is then selected based on these Q-values using a greedy strategy until a complete solution is formed. The GNN parameters are trained with the Deep Q-learning (DQN) algorithm to ensure accurate Q-value estimates. In related work, reference [14] synergizes Graph Neural Networks, DQN, and a greedy strategy for solution construction. This study employs Graph Convolutional Networks (GCNs) to model the graph structure, and the model was tested on large-scale instances of the Maximum Coverage Problem (MCP) at a 20k scale and the MVC problem at a 50k scale. Reference [15] utilizes Graph Neural Networks (GNNs) to estimate the probabilities of selecting edges in combinatorial optimization problems like the Traveling Salesman Problem (TSP). The GNN-based model outputs an adjacency matrix, and a feasible solution is constructed via beam search. The model is trained using supervised learning with datasets generated by solvers like LKH3 or Concorde and optimized using cross-entropy loss. However, its performance did not surpass traditional heuristic methods or the Pointer Network model.

Our choice of employing a Graph Convolutional Network is grounded in its demonstrated efficacy in analyzing and interpreting relational structures within data. GCNs have emerged as a powerful tool for capturing subtle patterns and dependencies, making them particularly suitable for the complex and heterogeneous nature of UC problems. The bipartite graph representation is selected for its ability to succinctly model the MILP structure, offering a clear visualization of the interplay between variables and constraints, which is crucial for understanding UC problems.

In our methodology, we employ graph neural networks to effectively forecast the instance most closely resembling a newly introduced instance, thereby leveraging the networks' capability to discern intricate patterns and relationships within the data.

## *C. Learning for UC Similarity*

In this section, as depicted in Fig. 2, we unveil our encompassing framework designed for the adaptive optimization of hyperparameters in solving unit commitment challenges via the MindOpt solver. Our strategy is underpinned by meticulous observations, which have prompted us to adopt the time expenditure associated with the solver's default parameter configuration as the definitive performance metric for each case study.

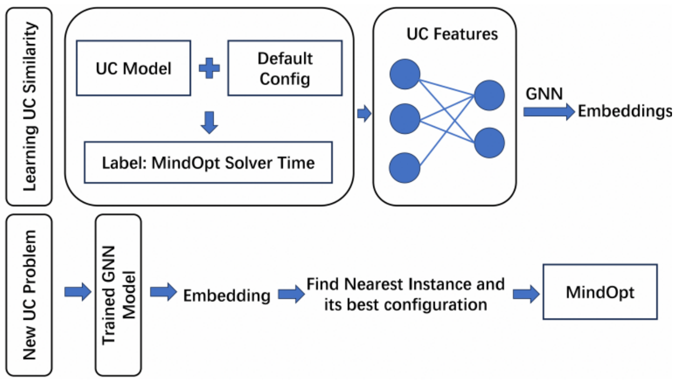

Figure 2. Framework of the Learning Approach

For each individual instance, we first capture its distinctive features as a bipartite graph, utilizing Ecole's readily available observation function [16]. These features are subsequently encoded through the application of a Graph Convolutional Network (GCN). During the training phase, the model is fine-tuned for regression tasks.

Upon completion of the training, for each emerging instance, we initially construct its bipartite graph representation and then deploy our trained model to predict its performance. With the predictive insight in hand, we identify the most akin historical instance to the new one and adopt its optimal parameter set as the hyperparameters for the new instance, thereby tailoring the solver's configuration to the unique characteristics of each problem at hand.

## IV. Experiments

In this section, we conduct numerical experiments to evaluate the efficacy of our novel learning approach for hyperparameter configuration in solving unit commitment problems via mixed integer linear programming solvers.

We begin by detailing the experimental setup, including dataset selection, config selection of MindOpt, performance metrics, and the computational environment. Subsequent results demonstrate the superiority of our approach in both solution quality and computational efficiency.

### *A. Setup*

Data: In the dataset subsection of our numerical experiments, we have selected the IEEE 30-bus system provided by the MATPOWER package as our primary test case for analysis [17]. The UnitCommitment.jl package offers comprehensive data for the IEEE 30-bus system for the entire year of 2017 [18]. This system's representativeness allows us to ensure that our findings are generalizable to a wide array of real-world power system scenarios. This dataset includes the necessary parameters and configurations to simulate realistic power system operations and unit commitment scenarios. For our experimental evaluation, we have utilized the full-year dataset available and have randomly sampled a subset of 10 days worth of .mps files to serve as our test set. The chosen days are representative of different load and generation conditions, providing a robust and thorough assessment of our proposed hyperparameter tuning approach for mixed integer linear programming solvers within the context of unit commitment problems.

**Config Selection:** In the numerical experiments conducted, we focus on the analysis of four key parameters within the MindOpt solver: Presolve, MIP/LevelCuts, MIP/LevelHeurs, and MIP/LevelProbing. The Presolve parameter dictates the activation state of the presolver technique. The MIP/LevelCuts parameter determines the aggressiveness of the cutting plane module, with higher values indicating greater assertiveness. Similarly, the MIP/LevelHeurs parameter sets the intensity of the heuristic search module, where larger values correspond to more vigorous search strategies. Lastly, the MIP/LevelProbing parameter adjusts the probing module's level of aggression, with increased values promoting more exhaustive probing efforts.

**Instance-wise Config Storage:** In the training phase, we exhaustively evaluated all possible hyperparameter combinations for the MindOpt solver, subsequently archiving the optimal configuration for each instance in our database. Our analysis revealed that distinct instances favored unique configurations, underscoring the variability in optimization requirements as presented in Table I.

TABLE I. Instance and Its Corresponding Optimal Config

| Instance | Presolve | LevelCuts | LevelHeurs | LevelProbing |
|---|---|---|---|---|
| 2017-01-18.mps | 0 | 0 | 0 | 1 |
| 2017-02-04.mps | 0 | 0 | 1 | 2 |
| 2017-06-05.mps | 0 | 0 | 0 | 2 |
| 2017-11-20.mps | 0 | 0 | 0 | 0 |

**Model Architecture and Training**: In the design of our graph neural network architecture, we drew inspiration from the concept proposed in [1]. In our experiment, the architecture of our model comprises four graph convolutional layers, followed by a max pooling layer and an attention pooling layer, culminating in a final linear layer for output. The comprehensive structure of the model is depicted in Fig. 3. During the training phase, we configured the batch size to 50 and set the number of training epochs at 20. The loss curve, which charts the progression of the training loss across epochs, is illustrated in Fig. 4.

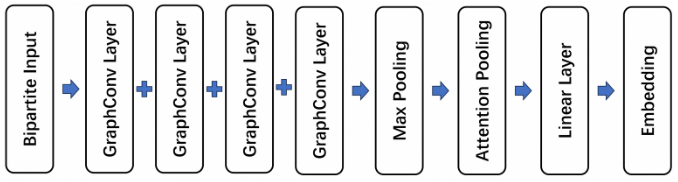

Figure 3. Model Architecture

To facilitate performance comparison, we have selected the execution time of the MindOpt solver as our primary metric. The Mipgap of all instances is set as 0.0001. All experimental evaluations are carried out on a machine equipped with an Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz, running Ubuntu 20.04, and utilizing 8 processing cores.

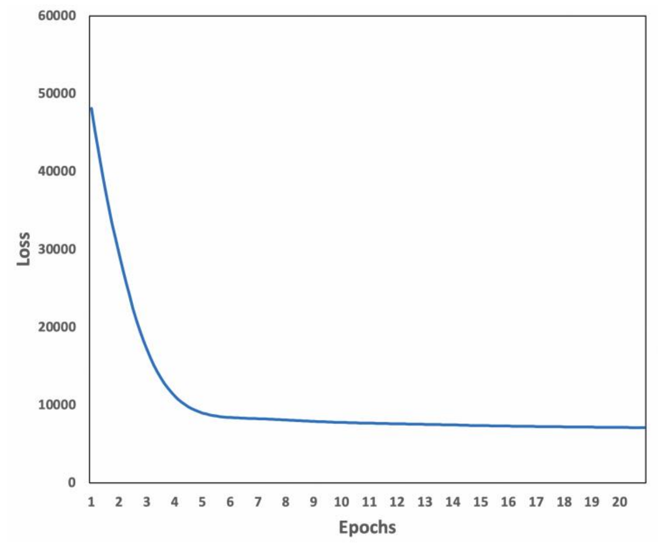

Figure 4. Training Loss Curve

### *B. Experimental Results*

We employ the trained model to obtain the optimal hyperparameter configuration for each test instance, which is then benchmarked against the average running time derived from all possible hyperparameter combinations. Table II illustrates how our learning-based approach effectively selects suitable parameters for new instances, demonstrating the method's efficiency and robustness in hyperparameter selection. Specifically, when new instance come, we construct embedding and find its corresponding instance with optimal hyperparameter setting. We then compare the execution time of our learning-based method against the average time taken by all potential hyperparameter combinations. The findings indicate that our approach achieves a roughly 1.8-fold speed increase over the average runtime.

TABLE II. MINDOPT RUNNING TIME ON TEST CASES

| New Instance | Corresponding Instance | Learning | Average |
|---|---|---|---|
| 2017-01-04.mps | 2017-01-18.mps | 1.14s | 1.47s |
| 2017-02-02.mps | 2017-06-05.mps | 12.1s | 30.1s |
| 2017-03-06.mps | 2017-02-04.mps | 7.55s | 9.98s |
| 2017-04-16.mps | 2017-08-02.mps | 4.91s | 9.38s |
| 2017-05-22.mps | 2017-01-06.mps | 12.5s | 24.0s |
| 2017-06-11.mps | 2017-07-05.mps | 0.75s | 0.82s |
| 2017-07-18.mps | 2017-06-28.mps | 0.79s | 0.85s |
| 2017-08-15.mps | 2017-11-20.mps | 4.30s | 7.40s |
| 2017-09-02.mps | 2017-09-12.mps | 63.1s | 109.0s |
| 2017-09-21.mps | 2017-12-05.mps | 6.32s | 11.9s |

## V. CONCLUSION

In this paper, we have introduced a novel learning approach aimed at optimizing the configuration of hyperparameters for mixed integer linear programming (MILP) solvers tackling unit commitment problems. Unlike generic hyperparameter tuning methods, our approach tailors solver configurations to the unique characteristics of each UC instance, leveraging a deep understanding of the problem's underlying structure. The essence of our methodology lies in the utilization of a bipartite graph representation of the MILP structure, coupled with the deployment of a Graph Convolutional Network (GCN) for the instance-wise tuning of solver parameters. Results demonstrated not only a significant improvement in solver performance but also underscored the potential for substantial reductions in computation time, reinforcing the practical applicability of our method in real-world unit commitment scenarios. Our findings contribute to the field by offering a new direction for optimization solver configuration, particularly for complex and dynamic problems such as UC in power systems.

Looking ahead, we identify several impactful research directions that could further propel the field forward. Incorporating additional physical and operational insights into the bipartite graph representation could potentially unveil more sophisticated patterns and relationships, thereby refining the predictive accuracy of our GCN model. Additionally, developing strategies to efficiently navigate the vast space of possible hyperparameter configurations, such as through strategic sampling or configuration embeddings, could lead to even more optimized solver settings.

## ACKNOWLEDGMENT

The work was supported by State Grid Corporation of China Technical Funding under Grant No. 5700-202358590A-3-2-ZN.

## REFERENCES

[1] A. Hosny and S. Reda, "Automatic MILP solver configuration by learning problem similarities," Annals of Operations Research, 2023.

[2] M. Zhang, W. Yin, M. Wang, Y. Shen, P. Xiang, Y. Wu, L. Zhao, J. Pan, H. Jiang, and K. Huang, "MindOpt Tuner: Boost the performance of numerical software by automatic parameter tuning," arXiv preprint arXiv:2307.08085, 2023.

[3] P. Chen, S. Liu, C. Shi, B. Hooi, B. Wang, and X. Cheng, "NeuCast: Seasonal neural forecast of power grid time series," in International Joint Conference on Artificial Intelligence, 2018.

[4] D. Pukhkaiev and U. Assmann, "Parameter tuning for self-optimizing software at scale," arXiv preprint arXiv:1909.03814, 2019.

[5] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," arXiv preprint arXiv:2007.15745, 2020.

[6] Y.-Q. Hu, Z. Liu, H. Yang, Y. Yu, and Y. Liu, "Derivative-free optimization with adaptive experience for efficient hyper-parameter tuning," in European Conference on Artificial Intelligence, 2020.

[7] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter, "Smac3: A versatile bayesian optimization package for hyperparameter optimization," The Journal of Machine Learning Research, vol. 23, no. 1, pp. 2475–2483, 2022.

[8] I. Himmich, E. M. Er Raqabi, N. El Hachemi, I. El Hallaoui, A. Metrane, and F. Soumis, "MPILS: An automatic tuner for MILP solvers," Computers & Operations Research, vol. 159, p. 106344, 2023.

[9] M. Sun, T. Li, and W. Yin, "MindOpt Adapter for CPLEX benchmarking performance analysis," arXiv preprint arXiv:2312.13527, 2023.

[10] S. Kadio ̆glu, Y. Malitsky, M. Sellmann, and K. Tierney, "ISAC - Instance-Specific Algorithm Configuration," in European Conference on Artificial Intelligence. IOS Press, 2010, pp. 751–756.

[11] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming," in RCRA workshop on experimental evaluation of

algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI), 2011, pp. 16–30.

[12] Z. Geng, X. Li, J. Wang, X. Li, Y. Zhang, and F. Wu, “A deep instance generative framework for milp solvers under limited data availability,” Advances in Neural Information Processing Systems, vol. 36, 2024.

[13] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, “Learning combinatorial optimization algorithms over graphs,” Advances in neural information processing systems, vol. 30, 2017.

[14] S. Manchanda, A. Mittal, A. Dhawan, S. Medya, S. Ranu, and A. Singh, “Learning heuristics over large graphs via deep reinforcement learning,” arXiv preprint arXiv:1903.03332, 2019.

[15] A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna, “A note on learning algorithms for quadratic assignment with graph neural networks,” stat, vol. 1050, p. 22, 2017.

[16] A. Prouvost, J. Dumouchelle, L. Scavuzzo, M. Gasse, D. Ch´etelat, and A. Lodi, “Ecole: A gym-like library for machine learning in combinatorial optimization solvers,” arXiv preprint arXiv:2011.06069, 2020.

[17] R. D. Zimmerman, C. E. Murillo-S´anchez, and R. J. Thomas, “Matpower: Steady-state operations, planning, and analysis tools for power systems research and education,” IEEE Transactions on power systems, vol. 26, no. 1, pp. 12–19, 2010.

[18] A. S. Xavier, A. M. Kazachkov, O. Yurdakul, and F. Qiu, “Unitcommitment. jl: A julia/jump optimization package for security-constrained unit commitment (version 0.3),” JuMP Optimization Package for Security-Constrained Unit Commitment (Version 0.3), Zenodo, 2022.