Exact solving scheduling problems accelerated by graph neural networks

Jana Juros*, Mario Brcic**, Mihael Koncic *** and Mihael Kovac**
* Student at University of Zagreb Faculty of Electrical Engineering and Computing, Zagreb, Croatia
** University of Zagreb Faculty of Electrical Engineering and Computing, Zagreb, Croatia
*** KONZUM plus d.o.o., Zagreb, Croatia
jana.juros@fer.hr, mario.brcic@fer.hr, mihael.koncic@konzum.hr, mihael.kovac@fer.hr

Abstract - Scheduling is a family of combinatorial problems where we need to find optimal time arrangements for activities. Scheduling problems in applications are usually notoriously hard to solve exactly. Existing exact solving procedures, based on mathematical programming and constraint programming, usually make manually-tuned heuristic choices. These heuristics can be improved by machine learning. In this paper, we apply the graph convolutional neural network from the literature on speeding up general branch&bound solver by learning its branching decisions. We test the augmented solver on job-shop scheduling problems and specific delivery scheduling problems in the supply chain of a local retailer. We get promising results and point to possible improvements. We discuss the interesting question of how much we can accelerate solving NP-hard problems in the light of the known limits and impossibility results in AI.

Keywords - combinatorial optimization; machine learning; job-shop scheduling problem; delivery scheduling; supply-chain; graph-convolution neural network; branch-and-bound algorithm; mixed-integer linear programming

I. INTRODUCTION

Combinatorial optimization (CO) is a subfield of mathematical optimization which aims to find the best decisions in problems with a big but finite set of alternatives. Application areas are numerous, such as supply chain management, project management, and vehicle routing. The problems are usually solved on a computer. It was immediately shown that such problems, unless having a special structure, are problematically complex for solving. Computational complexity has put most of such problems in a class of NP-hard problems for which the execution time takes too long for practical purposes. The naive general solving method would try to explicitly enumerate all options and select the best one. Current algorithmic approaches fall into two categories: exact methods of implicit enumeration over graphs and heuristic methods. The former are general and have solution quality guarantees, but their execution most often takes too long for practical purposes. Heuristic methods generally have no guarantees, but they give good-enough solutions in a realistic time.

Exact general methods fall into two general subcategories: approaches based on branch&bound (B&B) and constraint satisfaction methods. Neither method dominates the other. Among B&B-based approaches over mixed-integer linear programming (MILP), the most commonly used model in practice, Gurobi shows the best performance among commercial tools. SCIP is the best performing open-source solver [1]. Among tools for constraint satisfaction, CP-SAT from Google's OR-Tools has the best performance [2].

Heuristic approaches are divided into the following subcategories: constructive heuristics, metaheuristics, and hyperheuristics. We have included the category of improvement heuristics, commonly used in the literature, in metaheuristics by the principle of parsimony.

Exact methods, constructive heuristics, and metaheuristics search for the solution in the solution space. All of them depend on components made manually by human experts, which is often an expensive and time-consuming process. These components generally do not effortlessly transfer to the new types of problems. Namely, even exact methods use heuristic submethods in their decisions (such as branching variable-selection rules and selecting nodes for evaluation), which are not generally optimal.

Hyperheuristics, however, search the space of heuristics. Currently, the most dominant hyperheuristic approach is genetic programming. Such methods automatically find heuristics with good general performance on diverse problem-set. They mostly use features made by experts, and they use evolutionary methods for optimization, i.e. 0-order optimization methods that do not use the gradient information [3].

Lately, using deep neural approximators over combinatorial constructs has proven to be powerful in the domain of supervised learning in the problems of natural language processing with applications in computer programming [4] and mathematics [5]. That has resulted in substantial improvements in both research and applications. The successes above show the potential and capacity of such architectures to adequately approximate complex combinatorial structures in order to achieve necessary functionalities.

Moreover, the additional potential is demonstrated in using deep reinforcement learning (DRL) for attaining superhuman performance in games of chess, Go [6], Starcraft 2 [7], and Dota 2 [8]. All of the mentioned have huge combinatorial spaces over which action sequences are optimized. The main characteristic of deep neural approaches is the possibility of automatic discovery of specialized features that lead to better problem-solving capabilities and the reduced necessity for domain knowledge. As these approaches use 1-order optimization

methods (i.e., they use information about gradient), such methods are an order of magnitude faster than approaches based on 0-order counterparts [9]. Methods based on deep reinforcement learning achieved the most noticeable results over impressively large combinatorial problems with single objective (inherently or scalarized) [6]–[8], while control methods based on evolutionary approaches have lagged behind their successes due to lower sample efficiency [10], [11].

Additionally, current optimization methods treat problems as independent situations. However, often in practice, only related problems from the narrow domain are solved. Such a narrow domain can be the same family of problems or even a narrower domain where the problems share structure and differ only in the parameter values. For such settings, algorithms that could learn from the previous experience could, on average, find better solutions faster.

In this paper, we shall take two scheduling problems modeled as MILPs:

1. job shop scheduling problem (JSSP) – a canonical problem in machine scheduling, and
2. real-world delivery scheduling problem (DSP) in supply chain belonging to Konzum plus [12],

and we shall apply the method from the literature [13] to test the acceleration that could be gained on top of the exact branch&bound solver SCIP. We use the data generated from the runs of SCIP solver to do the imitation learning on graph convolutional neural network in order to learn a faster, high-quality branching policy.

This paper is structured as follows. We cover related work in section II, present our problem setting, results and elaborate upon them in section III. In section IV, we give our view on the possible future developments in the field of combinatorial optimization augmented with machine learning. Finally, in section V we shall give our conclusion.

II. Related work

Authors in [14] argue for the hybridization of combinatorial optimization approaches with machine learning in order to achieve better average performance over specific problem sets. There are several surveys into that area [15]–[22].

Recent works show the potential of using statistical, machine learning methods such as deep approximators when used in tandem with exact general solvers. The potential of approaches based on supervised learning in exact optimization on MILP was demonstrated in [13]. Graph convolutional neural networks were used to achieve improvement over other statistical approaches, as well as coded heuristics in SCIP solver. Qu et al. [23] used DRL to further improve the performance of the branching policy. The total B&B performance was further improved in [24] by the addition of neural diving to neural branching which bettered even the Gurobi solver. In [25], authors improve exact general solvers based on constraint satisfaction by using graph neural networks and deep reinforcement learning. Authors in [26] have used the DRL hyperheuristic for metric traveling salesman and knapsack problems, and for heterogenous vehicle routing problems [27]. The machine-learning model learned high-quality column selection for column generation procedure in [28] with a demonstration on vehicle routing problem with time windows.

Maragno et al. [29] established a methodology for mixed-integer optimization with learned constraints. Predict-and-optimize approaches in [30] used machine learning to learn the parameters of hard combinatorial problems while considering the effect of modeling errors on decisions. Subsequently, the problem was solved and achieved better results than classical ML error functions which only focus on achieving the model's precision on a certain dataset. Reinforcement learning was used in [31] to find dispatch policy for permutation flow shop scheduling with multiple lines.

Amazon and Alibaba, two of the world's greatest e-commerce companies with complex supply chains, have shown interest in fast hyperheuristic deep reinforcement learning approaches to solve combinatorial optimization problems like bin-packing [32], [33]. Authors in [32] have published basic baseline solutions for canonical combinatorial problems in supply chains with the motivation to advance the work in that area. So far, only the basic exploratory research and implementations have been made, and they show the potential of the DRL hyperheuristic approach.

III. Scheduling problems

In this section, we present two scheduling problems that we focused on – the job shop scheduling problem (JSSP) and the real-world supply chain management problem obtained from Konzum plus data. Next, we describe B&B algorithm and parametrization of branching policy. Finally, we describe the process of generating data and our model – a graph convolution neural network.

A. Job shop scheduling problem

The job-shop scheduling problem (JSSP) is an optimization problem in which jobs are assigned to resources at a given time. The standard version is as follows: suppose we have a finite set J of n jobs and a finite set M of m machines. For each job j in J, we have a list of machines $(o_1^j, \dots, o_h^j, \dots, o_m^j)$ that represents the processing order of j through machines. Additionally, for every job j and machine i, a non-negative integer p_{ij} is assigned, which represents the processing time of job j on machine i. Each machine can process a maximum of one job at a time and when the job starts on a particular machine, its processing must be completed without interruption. The goal is to find a schedule of processing J on M that minimizes the processing time. Minimizing processing time for JSSP is an NP difficult problem for $n \geq 3$ and $m \geq 2$ [34]. JSSP can be formulated as a MILP problem. According to [34], Manne's (disjunctive) model [35] is the best formulation, so we decided to use that model. In the rest of the paper, we shall refer to sizes

of JSSP problems as *nxm*, that is (*number of jobs x number of machines*).

B. Supply chain management problem

The problem instances which we use for training, testing, and evaluating the network are real-life instances of supply chain management problems [12], collected from a network of Konzum stores. Specifically, the data includes delivery needs for 420 Konzum branches, taking into account utilization of transport and warehouses, and product shelf life. A solution to the problem consists of weekly delivery patterns, i.e. the baseline schedule taken as the basis for schedules in all weeks for a certain period. The problem has 40,000 (mostly binary) variables and 46,000 constraints.

C. Branch&bound algorithm

B&B algorithm implements a "divide-and-conquer" algorithm that can be represented by a search tree in which LP relaxation of the problem is calculated at each node. If the relaxation is infeasible, or if the solution of the relaxation is naturally MILP feasible (integrality requirements are respected), the node does not need to be expanded. Otherwise, there exists at least one variable, among those supposed to be an integer, which has a non-integer value in the LP solution and that variable can be chosen for branching.

The iterative procedure of the algorithm requires two sorts of sequential decision making:

1. **Node selection** - selection of the next node to evaluate

2. **Variable selection** - selection of the variable by which the node search space will be partitioned i.e. variable by which the tree will be further branched

D. Application of graph convolutional neural network in the B&B algorithm

In this paper, we will use a graph convolutional neural network to speed up the B&B algorithm in the SCIP solver. More precisely, we will focus on the variable selection, also known as the branching problem, which is the core of the B&B paradigm. We will try to adopt a strategy of imitation learning to learn a fast approximation of strong branching. Full strong branching is a high-quality, but expensive branching rule which is consistently resulting in the smallest B&B trees [13]. It does so by computing the expected bound improvement for each variable that is a candidate for node branching, which unfortunately requires two LP solutions for each candidate. In practice, strong branching is not used at every node. Instead, modern solvers rely on hybrid branching. Hybrid branching uses strong branching at the beginning of the solving process and gradually transitions to simpler heuristics such as the conflict score, the pseudo-cost, or a combination of the two.

Our goal is to make a model (graph convolutional neural network) that imitates strong branching in SCIP for a particular class of problem. We will achieve this by solving numerous problems of the same class in SCIP and simultaneously "recording" pairs (state, action). That is, for states in the algorithm we will observe which variable will be chosen for node expanding by full strong branching (FSB) policy. We will train and test the network with obtained pairs and try to achieve that the model imitates the mentioned policy. When the model is trained, we will integrate it into SCIP as a new branching policy, which works about as well as FSB, but significantly faster. Therefore, by using statistical caching, interpolations, and extrapolations in neural networks, we will try to speed up the solving algorithm.

E. State parametrization of the B&B algorithm

Because MILP problems consist of variables and constraints related to those variables, the natural representation of state is shown in the form of a bipartite graph (Figure 1). Bipartite graph consists of constraint nodes and variable nodes. If a constraint refers to a variable, then corresponding nodes in the graph are connected via edge. Furthermore, suppose that in the state of the solver (B&B algorithm) there are m constraints, each of which will have e features, and n variables, each of which has d features. Then the constraint nodes are represented by a constraint feature matrix $\boldsymbol{C} \in \mathbb{R}^{m \times c}$, and the variable nodes are represented by a variable feature matrix $\boldsymbol{V} \in \mathbb{R}^{n \times d}$. Finally, the edges are represented by an edge feature matrix $\boldsymbol{E} \in \mathbb{R}^{m \times n \times e}$. Thus, we parameterize the state s_t of the B&B process in moment t as a bipartite graph $(\boldsymbol{G})$ with the node features matrices and edge features matrix $(\boldsymbol{G}, \boldsymbol{C}, \boldsymbol{E}, \boldsymbol{V})$.

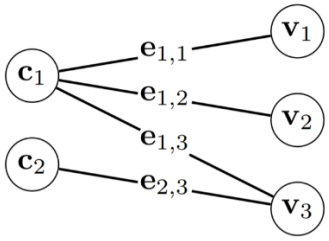

Figure 1 State of the solver - bipartite graph with constraint and variable nodes *[13]*

F. Parametrization of variable selection in B&B algorithm

The variable selection policy $\pi_0(a|s_t)$ is parameterized as a graph convolutional neural network shown in *Figure 2*. We wrote the network model and training procedure based on the code in [13]. Our model takes the mentioned bipartite representation of the state $s_t = (\boldsymbol{G}, \boldsymbol{C}, \boldsymbol{E}, \boldsymbol{V})$ as input and performs one graph convolution, in form of two intertwined semi-convolutions.

Namely, due to the bipartite structure of the input graph, our graph convolution can be divided into two consecutive passes and thus, perform the whole convolution in an optimized way.

In a convolution, we use 2-layer perceptron with *ReLU* as an activation function. Passing through the graph convolution layer, we obtain a bipartite graph with the same topology as the input, but with potentially different node characteristics, so that each node now contains the accumulated information from its neighbors. The required variable selection policy is obtained by discarding the constraint nodes and applying the final 2-layer perceptron on variable nodes, combined with *softmax* activation function to obtain the probability distribution over the variables that are branching candidates.

Finally, we filter branching candidates (obtained from the SCIP state) and refresh the loss function depending on whether the choice of our model coincides with the choice of the strong branching policy.

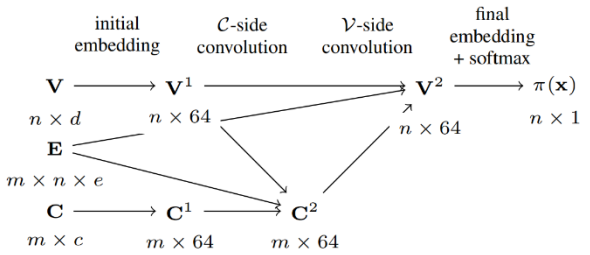

Figure 2 Representation of the layers in graph convolutional neural network used for parameterization of branching policy $\pi_0(a|s_t)$ *[13]*

G. Methodology

We present a compendious summary of the whole procedure:

1. Data collection – collecting train and validation samples by initiating solving of MILP instances in SCIP and recording the parameterized pairs (state, action) which represent a policy of FSB every five solving steps. We must emphasize that FSB is a quite slow procedure on our JSSP and DSP instances since these problems have thousands of variables.
    a. JSSP – we have used 10x10 randomly created JSSP instances to generate 100,000 training, 20,000 validation, and 20,000 test samples of the SCIP state. We used separate JSSP instances for each dataset.
    b. DSP – we have used 36 problem instances and separated them into 24 for training, 6 for validation, and 6 for testing. Since each problem has 40,000 variables, it is much too big to run FSB policy for extracting solver states. Hence, we decompose each of the problems into 10 problems with 2000 variables.
2. training - using training and validation samples to train our GCNN model to imitate FSB policy. We have used a batch size of 32 for training and 128 for evaluation, Adam optimizer, and cross-entropy loss. We stopped training after 10 successive epochs without improvement on the validation set or 312 epochs at the latest.
3. testing – using new test samples on our trained network to assess the accuracy of the model in selecting the branching variable
4. CO evaluation – solving new MILP instances in SCIP using our model for variable selection and comparing solving speed with "ordinary" SCIP
    a. for JSSP we have used 20 instances for each of easy (7x7), medium (10x10), and hard (11x11) problem sizes.
    b. For DSP we have used 34 feasible decomposed problem instances from the test set (the remaining 26 were infeasible) for the **small**-scale test. Later we also test the generalization on 6 original **large** instances (prior to decomposition). We have used the gap of 0.8% and timeout of 1h for large instances.

H. Results

Table 1 Model testing results

Dataset	acc@1	acc@3	acc@5	acc@10
JSSP	42.4	64.0	67.9	77.8
DSP	46.0	64.9	73.2	83.1

From testing results in Table 1, we can see that the model has partially learned to imitate the FSB policy. Column **acc@k** holds the value of proportion of situations where the correct choice was in top-*k* choices of our GCNN model. The precision is lower than for the selected problems in [13]. We will have to evaluate the model to see whether it will be able to speed up solving of problem instances – and this depends on the performance of the chosen variables when branching policy unavoidably misses the FSB choice. However, the performance of the node is not something the model was optimized for, but only the selection of top choice.

Table 2 Model evaluation results (* wins due to timeout)

Dataset	Difficulty	Branching policy	Solving time	# of best	Number of nodes
JSSP	Easy	RPB	1.0 ± 17%	14	173 ± 31%
		GCNN	7.1 ± 55%	6	97 ± 45%
	Medium	RPB	11.8 ± 44%	17	3198 ± 39%
		GCNN	745.6 ± 52%	3	17031 ± 45%
	Hard	RPB	77.6 ± 2%	19	27703 ± 88%
		GCNN	3113.6 ± 1087%	1	17228 ± 31%
DSP	Decomp.	RPB	15.0 ± 96%	12	639 ± 194%
		GCNN	32.0 ± 207%	13	2977 ± 257%
	Large	RPB	316 ± 50%	2*	277 ± 79%
		GCNN	1319 ± 134%	2	4785 ± 148%

We can see in Table 2 that the algorithm with a default reliable pseudo-cost branching (RPB) policy tends to perform faster than our approach, which can be attributed

to running GCNN on CPU for JSSP [36]. Therefore, we shall focus on the number of processed nodes as a more reliable indicator. From that aspect, we can see that GNN indeed reduces the number of processed nodes for smaller JSSP instances. However, with the growing number of variables, branching errors accumulate and tend to produce worse subtrees – missed choices are not of high quality. GCNN calculations, unlike RPB, are trivially parallelizable. In the DSP problem, we ran GCNN on GPU. Out of used 34 CO-evaluation instances, our algorithm solved the problem faster in 13 of the DSP instances, while on 9 it was equal to RPB. However, sometimes when GCNN mis-performed, it did so extremely. On large instances of DSP, GCNN model won twice (40%,173% speedup) and tied twice. However, it also *led to timeout* twice! We can also observe a very large variance in solving time and number of nodes in both algorithms. That happens because the problem instances are very diverse in the terms of solving difficulty. Also, the distribution of the number of nodes and the solving times is skewed to the right due to the small number of problems that take a very long time to solve. Those problems significantly increase average and variance in the number of nodes and solving times.

IV. Discussion

There is a noticeable difference in performance on JSSP and DSP. There are two factors at play: DSP has a simpler problem structure, and the problem instances we have used have a smaller variety since they are created on top of the identical distribution network from data through 36 weeks. Contrarily, we have generated JSSP instances uniformly. We have also seen that surrogate ML measures do not work well in all circumstances. Cross-entropy loss does not capture well the performance of the branching since its missed choices do not have graceful performance degradation. As JSSP and DSP have „tighter“ feasibility than the problems mostly tackled in the literature (i.e. set cover, combinatorial auction, maximal anticlique, capacitated facility location), using diving heuristic such as in [24] might be a promising direction for improvement.

Currently, exact solvers do not learn from experience. That means that if we input the same instance twice in succession into a solver, it will behave without memory and solve them independently. Trivially, it could simply memorize the shortest tree path to the solution and repeat it for all the problems with small perturbations to the original problem for potentially great speedup. If some learning algorithm could have the capacity to learn the partitioning of the appropriately reduced problem family space, significant gains could be achieved on average. The limits to how much can solving NP-hard problems be accelerated with machine learning hyperheuristics is an open and highly interesting question, especially under the known limits of AI [37] such as no-free-lunch theorems. Graph neural networks seem like a natural fit for solving these problems since some of the models are provably robust on countable domains [38], but many of the architectures rely on biases to achieve good results [39]. Explainability methods might help in debugging and crafting better graph-neural network architectures that would contain more application-specific biases [40]–[42]. Explainability was already used for insights into combinatorial optimization algorithms such as for project scheduling [43], [44], knapsack problems, facility location, etc. [45].

Our work is among the first ones that applied ML methods to scheduling problems and on real-world problems. Applications in scheduling, packing, and routing problems for supply chains, combinatorial design [46], and compiler optimizations [47] should be tackled with these methods. Finding CO-focused learning metrics that are efficient and align well with the solving performance is of the greatest interest.

V. Conclusions

Coupling combinatorial optimization solvers with machine learning has many peculiarities that are not usually encountered – for example very costly data collection and computational hardness of the underlying problem.

We have adapted the solution from the literature and applied it to two scheduling problems: job shop scheduling problem and real-world delivery scheduling in the supply chain. This is one of the first applications of such methods in the area of scheduling. We got promising results with successes on small instances of JSSP and on real-world delivery scheduling with good generalization to larger instances for the latter. We have seen that the **sampling distribution of problems greatly matters** for achievable performance. High entropy distributions incur big penalties on performance. Also important is that CO-focused metrics are used for the learning task, instead of the classification objective which is an ML-based surrogate that is ill-fitted to these problems.

Literature

[1] H. Mittelman, “MILP benchmarks,” *The MIPLIB2017 Benchmark Instances*. http://plato.asu.edu/ftp/milp.html (accessed Feb. 06, 2022).

[2] “MiniZinc - Challenge 2021.” https://www.minizinc.org/challenge2021/results2021.html (accessed Feb. 06, 2022).

[3] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, “Automated Design of Production Scheduling Heuristics: A Review,” *IEEE Trans. Evol. Comput.*, vol. 20, no. 1, pp. 110–124, Feb. 2016, doi: 10.1109/TEVC.2015.2429314.

[4] Y. Li, D. Choi, and J. Chung, “Competitive programming with AlphaCode,” Feb. 02, 2022. Accessed: Feb. 05, 2022. [Online]. Available: https://storage.googleapis.com/deepmind-media/AlphaCode/competition_level_code_generation_with_alphacode.pdf

[5] S. Polu, J. M. Han, K. Zheng, M. Baksys, I. Babuschkin, and I. Sutskever, “Formal Mathematics Statement Curriculum Learning,” *ArXiv220201344 Cs*, Feb. 2022, Accessed: Feb. 05, 2022. [Online]. Available: http://arxiv.org/abs/2202.01344

[6] D. Silver *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018, doi: 10.1126/science.aar6404.

[7] O. Vinyals *et al.*, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, Art. no. 7782, Nov. 2019, doi: 10.1038/s41586-019-1724-z.

[8] OpenAI *et al.*, “Dota 2 with Large Scale Deep Reinforcement Learning,” Dec. 2019, Accessed: Feb. 26, 2020. [Online]. Available: https://arxiv.org/abs/1912.06680v1

[9] J. C. Duchi, M. I. Jordan, M. J. Wainwright, and A. Wibisono, “Optimal Rates for Zero-Order Convex Optimization: The

Power of Two Function Evaluations," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2788–2806, May 2015, doi: 10.1109/TIT.2015.2409256.
[10] N. Müller and T. Glasmachers, "Challenges in High-Dimensional Reinforcement Learning with Evolution Strategies," in *Parallel Problem Solving from Nature – PPSN XV*, Cham, 2018, pp. 411–423. doi: 10.1007/978-3-319-99259-4_33.
[11] A. Y. Majid, S. Saaybi, T. van Rietbergen, V. Francois-Lavet, R. V. Prasad, and C. Verhoeven, "Deep Reinforcement Learning Versus Evolution Strategies: A Comparative Survey," *ArXiv211001411 Cs*, Sep. 2021, Accessed: Apr. 16, 2022. [Online]. Available: http://arxiv.org/abs/2110.01411
[12] A. Alajković, M. Brcic, V. Ivandić, L. Bubalo, M. Koncic, and M. Kovač, "Delivery pattern planning in retailing with transport and warehouse workload balancing," *Rev.*, p. 12, Apr. 2022, doi: 10.13140/RG.2.2.23386.62402.
[13] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact Combinatorial Optimization with Graph Convolutional Neural Networks," *ArXiv190601629 Cs Math Stat*, Oct. 2019, Accessed: Feb. 22, 2020. [Online]. Available: http://arxiv.org/abs/1906.01629
[14] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, Apr. 2021, doi: 10.1016/j.ejor.2020.07.063.
[15] J. Kotary, F. Fioretto, P. Van Hentenryck, and B. Wilder, "End-to-End Constrained Optimization Learning: A Survey," *ArXiv210316378 Cs*, Mar. 2021, Accessed: Feb. 06, 2022. [Online]. Available: http://arxiv.org/abs/2103.16378
[16] Q. Cappart, T. Moisan, L.-M. Rousseau, I. Prémont-Schwarz, and A. Cire, "Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization," *ArXiv200601610 Cs*, Jun. 2020, Accessed: Feb. 05, 2022. [Online]. Available: http://arxiv.org/abs/2006.01610
[17] Y. Peng, B. Choi, and J. Xu, "Graph Learning for Combinatorial Optimization: A Survey of State-of-the-Art," *Data Sci. Eng.*, vol. 6, no. 2, pp. 119–141, Jun. 2021, doi: 10.1007/s41019-021-00155-3.
[18] Q. Wang and C. Tang, "Deep reinforcement learning for transportation network combinatorial optimization: A survey," *Knowl.-Based Syst.*, vol. 233, p. 107526, Dec. 2021, doi: 10.1016/j.knosys.2021.107526.
[19] A. Popescu *et al.*, "An overview of machine learning techniques in constraint solving," *J. Intell. Inf. Syst.*, 2021, doi: 10.1007/s10844-021-00666-5.
[20] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E.-G. Talbi, "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art," *Eur. J. Oper. Res.*, vol. 296, no. 2, pp. 393–422, 2022, doi: 10.1016/j.ejor.2021.04.032.
[21] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Comput. Oper. Res.*, vol. 134, p. 105400, Oct. 2021, doi: 10.1016/j.cor.2021.105400.
[22] Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković, "Combinatorial Optimization and Reasoning with Graph Neural Networks," Aug. 2021, vol. 5, pp. 4348–4355. doi: 10.24963/ijcai.2021/595.
[23] Q. Qu *et al.*, "An Improved Reinforcement Learning Algorithm for Learning to Branch," *ArXiv220106213 Cs Math*, Jan. 2022, Accessed: Feb. 07, 2022. [Online]. Available: http://arxiv.org/abs/2201.06213
[24] V. Nair *et al.*, "Solving Mixed Integer Programs Using Neural Networks," *ArXiv201213349 Cs Math*, Jul. 2021, Accessed: Feb. 06, 2022. [Online]. Available: http://arxiv.org/abs/2012.13349
[25] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro, "Improving SAT Solver Heuristics with Graph Networks and Reinforcement Learning," *ArXiv190911830 Cs*, Sep. 2019, Accessed: Feb. 26, 2020. [Online]. Available: http://arxiv.org/abs/1909.11830
[26] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural Combinatorial Optimization with Reinforcement Learning," *ArXiv161109940 Cs Stat*, Jan. 2017, Accessed: Feb. 05, 2022. [Online]. Available: http://arxiv.org/abs/1611.09940
[27] W. Qin, Z. Zhuang, Z. Huang, and H. Huang, "A novel reinforcement learning-based hyper-heuristic for heterogeneous vehicle routing problem," *Comput. Ind. Eng.*, vol. 156, p. 107252, Jun. 2021, doi: 10.1016/j.cie.2021.107252.
[28] M. Morabit, G. Desaulniers, and A. Lodi, "Machine-learning-based column selection for column generation," *Transp. Sci.*, vol. 55, no. 4, pp. 815–831, 2021, doi: 10.1287/trsc.2021.1045.
[29] D. Maragno, H. Wiberg, D. Bertsimas, S. I. Birbil, D. den Hertog, and A. Fajemisin, "Mixed-Integer Optimization with Constraint Learning," *ArXiv211104469 Cs Math Stat*, Nov. 2021, Accessed: Feb. 06, 2022. [Online]. Available: http://arxiv.org/abs/2111.04469
[30] J. Mandi, E. Demirovi?, P. J. Stuckey, and T. Guns, "Smart Predict-and-Optimize for Hard Combinatorial Optimization Problems," *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 02, Art. no. 02, Apr. 2020, doi: 10.1609/aaai.v34i02.5521.
[31] J. Brammer, B. Lutz, and D. Neumann, "Permutation flow shop scheduling with multiple lines and demand plans using reinforcement learning," *Eur. J. Oper. Res.*, vol. 299, no. 1, pp. 75–86, 2022, doi: 10.1016/j.ejor.2021.08.007.
[32] B. Balaji *et al.*, "ORL: Reinforcement Learning Benchmarks for Online Stochastic Optimization Problems," *ArXiv191110641 Cs Math*, Dec. 2019, Accessed: Feb. 26, 2020. [Online]. Available: http://arxiv.org/abs/1911.10641
[33] H. Hu, X. Zhang, X. Yan, L. Wang, and Y. Xu, "Solving a New 3D Bin Packing Problem with Deep Reinforcement Learning Method," *ArXiv170805930 Cs*, Aug. 2017, Accessed: Feb. 26, 2020. [Online]. Available: http://arxiv.org/abs/1708.05930
[34] W.-Y. Ku and J. C. Beck, "Mixed Integer Programming models for job shop scheduling," *Comput. Oper. Res.*, vol. 73, no. C, pp. 165–173, Sep. 2016, doi: 10.1016/j.cor.2016.04.006.
[35] A. S. Manne, "On the Job-Shop Scheduling Problem," *Oper. Res.*, vol. 8, no. 2, pp. 219–223, Apr. 1960, doi: 10.1287/opre.8.2.219.
[36] P. Gupta, M. Gasse, E. B. Khalil, M. P. Kumar, A. Lodi, and Y. Bengio, "Hybrid Models for Learning to Branch," *ArXiv200615212 Cs Math Stat*, Oct. 2020, Accessed: Feb. 07, 2022. [Online]. Available: http://arxiv.org/abs/2006.15212
[37] M. Brcic and R. V. Yampolskiy, "Impossibility Results in AI: A Survey," *ArXiv210900484 Cs*, Sep. 2021, Accessed: Feb. 06, 2022. [Online]. Available: http://arxiv.org/abs/2109.00484
[38] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?," *ArXiv181000826 Cs Stat*, Feb. 2019, Accessed: Feb. 04, 2022. [Online]. Available: http://arxiv.org/abs/1810.00826
[39] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond Euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017, doi: 10.1109/MSP.2017.2693418.
[40] F. K. Dosilovic, M. Brcic, and N. Hlupic, "Explainable artificial intelligence: A survey," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, May 2018, pp. 0210–0215. doi: 10.23919/MIPRO.2018.8400040.
[41] M. Juric, A. Sandic, and M. Brcic, "AI safety: state of the field through quantitative lens," Feb. 2020, Accessed: Feb. 22, 2020. [Online]. Available: https://arxiv.org/abs/2002.05671v1
[42] A. Krajna, M. Brcic, M. Kovac, and A. Sarcevic, "Explainable Artificial Intelligence: An Updated Perspective," presented at the under review, 2022.
[43] M. Brčić and D. Mlinarić, "Tracking Predictive Gantt Chart for Proactive Rescheduling in Stochastic Resource Constrained Project Scheduling," *J. Inf. Organ. Sci.*, vol. 42, no. 2, pp. 179–192, Dec. 2018, doi: 10.31341/jios.42.2.2.
[44] M. Brčić, M. Katić, and N. Hlupić, "Planning horizons based proactive rescheduling for stochastic resource-constrained project scheduling problems," *Eur. J. Oper. Res.*, vol. 273, no. 1, pp. 58–66, Feb. 2019, doi: 10.1016/j.ejor.2018.07.037.
[45] D. Bertsimas and B. Stellato, "The voice of optimization," *Mach. Learn.*, vol. 110, no. 2, pp. 249–277, Feb. 2021, doi: 10.1007/s10994-020-05893-5.
[46] M. Brčić and D. Kalpić, "Combinatorial testing in software projects," in *2012 Proceedings of the 35th International Convention MIPRO*, May 2012, pp. 1508–1513.
[47] M. Kovac, M. Brcic, A. Krajna, and D. Krleza, "Towards intelligent compiler optimization," presented at the under review, 2022. doi: 10.13140/RG.2.2.29644.49288.