# [Re] Exact Combinatorial Optimization with Graph Convolutional Neural Networks

**Audrey-Anne Guindon**
HEC Montreal
audreyannehebertguindon@gmail.com

**Lourdes Crivelli**
HEC Montreal
crivellilourdes@gmail.com

## Abstract

In this work we reproduce and corroborate the results presented in the paper "Exact Combinatorial Optimization with Graph Convolutional Neural Networks" submitted to the NeurIPS 2019 Conference [1]. We implement the published code, run the experiments and propose suggestions to improve the overall reproducibility of the paper. We expand the work by running the ablation study described in the paper on a new NP-Hard problem, the combinatorial auction problem. We conclude that the paper is reproducible and that the proposed method does reduce the solving time of the combinatorial auction problem. The code repository can be accessed through the following link: `https://github.com/audreyanneguindon/NeurIPS_2019`

## 1 Introduction

The paper "Exact Combinatorial Optimization with Graph Convolutional Neural Networks" [1] proposes an innovative approach to learning branch-and-bound variable selection policies by applying graph convolution neural network (GCNN) models to learn branch-and-bound variable selection for mixed-integer linear programs (MILP). The authors' objective is to learn a policy for variable selection in MILPs that leads to shorter computation time without relying on hard-coded heuristics. In order to achieve this, they train their model on an expert strong branching rule using behavioural cloning to learn variable selection policies that produce the smallest search trees. These policies are then tested on larger problem instances to prove the ability of the model to generalize.

The authors propose to encode the branching policies into a GCNN, which allows to exploit the natural bipartite graph representation of MILP problems, thereby reducing the need for manual feature engineering. GCNNs provide other advantages, they are well defined regardless of the input graph size, they will always produce the same outputs regardless of the order in which the nodes are presented, and their computational complexity is directly related to the density of the graph. These advantages provide improvements over other machine learning methods for learning branch-and-bound variable selection policies, by reducing the need for feature engineering and allowing the method to generalize to larger problem instances than those seen during training.

In the following reproducibility paper, our aim is to test the proposed GCNN model and determine whether we can achieve the same results as reported by the authors. To determine the reproducibility of the paper and confirm the authors' conclusions, the main questions we will answer are:

- Can we reproduce the reported results for the combinatorial auction problems with the given code?
- Is the proposed model, when trained on smaller problem instances, able to generalize to larger instances? Does it provide a substantial advantage in terms of computational time and/or accuracy?
- How can the code be further expanded to facilitate reproduction?

Preprint. Under review.

In Section 2, we review the related literature. In Section 3, we describe in detail the approach presented by the authors. In Section 4, we discuss the implementation of the code and provide insights and suggestions to assist researchers in implementing the model. In Section 5, we highlight the experiments' results and contrast them with the authors' work. Finally, we discuss our findings in Section 6.

## 2 Related work

Most NP-Hard computer problems (nondeterministic polynomial time) are examples of combinatorial optimization problems [1]. These types of problems, although difficult to solve, can be tackled by a broad range of algorithms that are able to find an optimal solution at the cost of exponential time complexity [2]. One such method involves formulating the problem as a mixed-integer linear program (MILP) and using the branch-and-bound method to find the optimal solution [3].

Branch-and-bound algorithms have been extensively studied, but there is still a lack of deep mathematical understanding of the decision process involved in selecting the variable by which to partition the node's search space [4]. The current literature has focused on computational studies of said process, and has recently been combined with machine learning [5, 6, 7]. However, these approaches rely heavily on problem-specific strategies [4]. This raises a concern for the authors, as the inability to generalize a policy impacts its application. The authors' propose to build a model that can be applied regardless of the problem formulation, and that is able to generalize to problem instances larger than those seen during training. In order to achieve this, their work focuses on the variable branching decision of the branch-and-bound method.

Building on previous research, which shows that the sequential decisions made by the branch-and-bound method can be assimilated to a Markov decision process [8, 9], the authors encode the states of the branch-and-bound process as a bipartite graph and use GCNNs to study the decisions made at each state. Khalil et al. [13] first proposed a GCNN model for learning greedy heuristics on several collections of combinatorial optimization problems defined on graphs, which provides evidence that GCNNs can effectively capture characteristics of combinatorial optimization problems [1]. By using GCNN, the researchers are able to leverage the natural bipartite graph representation of MILP problems to avoid manual feature engineering [1], which is one of the concerns the paper attempts to solve. Although there are numerous strategies for selecting a branching variable in the branch-and-bound method, strong branching was selected because it produces the smallest search trees [5].

Previous works [5, 6, 7] have used imitation learning to learn a substitute function to replicate the strong branching strategy at reduced computational time. For instance, Khalil et al. [5] and Hansknecht et al. [7] treat the task as a ranking problem and learn a partial ordering of the candidates produced by the expert, while Alvarez et al. [6] treat it as a regression problem and learn directly the strong branching scores of the candidates. In contrast to these imitation learning studies [10], the authors treat the task as a classification problem, and the proposed model is trained on the expert branching strategy using behavioral cloning, which does not rely on branching scores or ordering. The policy is selected by minimizing the cross-entropy loss [1]. The authors' paper differs from previous work by contrasting the results of their method against both state-of-the-art branching rules and related machine learning branchers [5, 6, 7]. They also evaluate their work on different NP-Hard combinatorial optimization problems, and demonstrate that their method is able to generalize to larger instances.

Our contribution through this reproduction paper will be to clarify the implementation of the model and corroborate the results reported. We also address reproducibility concerns and offer recommendations based on our experience and the insights we obtained during the reproduction.

## 3 Background

The authors assimilate the sequential decisions made during branch-and-bound to a Markov decision process [9]. The core problem of Markov decision processes is to find a policy for the agent, that is, a function $\pi(\mathbf{a}|\mathbf{s}_t)$ that specifies the action $\mathbf{a}_t$ that the agent will choose when in state $\mathbf{s}_t$.

If we consider the solver to be the environment, and the brancher the agent, then at decision $t$, the solver is in a state $\mathbf{s}_t$, and the brancher selects a variable $\mathbf{a}_t$ among all fractional variables $\mathcal{A}(\mathbf{s}_t) \subseteq \{1, \ldots, p\}$ according to a policy $\pi(\mathbf{a}|\mathbf{s}_t)$. A state $\mathbf{s}_t$ is comprised of the branch-and-bound tree with all past branching decisions, the best solution found so far, the LP solution of each node, and the current leaf node. Each episode in the branch-and-bound Markov decision process amounts to solving a MILP problem instance. The initial state corresponds to an instance being sampled, while final states mark the end of the optimization process. The probability of a trajectory $\tau = (\mathbf{s}_0, ..., \mathbf{s}_T) \in \mathcal{T}$ then depends on both the branching policy and the remaining components of the solver

$$p_\pi(\tau) = p(\mathbf{s}_0) \prod_{t=0}^{T-1} \sum_{\mathbf{a} \in \mathcal{A}(\mathbf{s}_t)} \pi(\mathbf{a}|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}).$$

Instead of using reinforcement learning to find a good branching policy, the authors use behavioural cloning to learn directly from an expert branching rule. The process involves running the strong branching rule on a collection of MILP training instances, from which a dataset of state-action pairs $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i^*)\}_{i=1}^N$ is recorded. Then, the policy is learned by minimizing the cross-entropy loss

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{(\mathbf{s}, \mathbf{a}^*) \in \mathcal{D}} \log \pi_\theta(\mathbf{a}^*|\mathbf{s}).$$

The state $\mathbf{s}_t$ of the branch-and-bound process at time $t$ is encoded as a bipartite graph with node and edge features $(\mathcal{G}, \mathbf{C}, \mathbf{E}, \mathbf{V})$. The intuitive idea underlying GNNs is that nodes in a graph represent concepts and edges represent their relationships. One side of the bipartite graph are nodes corresponding to the constrains of the MILP, one per row in the current LP relaxation, with $\mathbf{C} \in \mathbb{R}^{m \times c}$. The other side of the bipartite graph are nodes corresponding to the variables of the MILP, one per LP column, with $\mathbf{V} \in \mathbb{R}^{n \times d}$. The edge $\mathbf{e}(i, j) \in \mathcal{E}$ connects a constraint node $i$ and a variable node $j$ if the variable is involved in the constraint.

The authors parametrize the variable selection policy $\pi(\mathbf{a}|\mathbf{s}_t)$ as a graph convolutional neural network. The model takes as input the bipartite state representation $\mathbf{s}_t = (\mathcal{G}, \mathbf{C}, \mathbf{E}, \mathbf{V})$ and performs a single graph convolution, in the form of two interleaved half-convolutions. The graph convolution is broken down into two successive passes, one from variables to constraints and one from constraints to variables. These passes take the form

$$\mathbf{c}_i \leftarrow \mathbf{f}_\mathcal{C}\left(\mathbf{c}_i, \sum_j^{(i,j)\in\mathcal{E}} \mathbf{g}_\mathcal{C}(\mathbf{c}_i, \mathbf{v}_i, \mathbf{e}_{i,j})\right), \quad \mathbf{v}_j \leftarrow \mathbf{f}_\mathcal{V}\left(\mathbf{v}_j, \sum_i^{(i,j)\in\mathcal{E}} \mathbf{g}_\mathcal{V}(\mathbf{c}_i, \mathbf{v}_i, \mathbf{e}_{i,j})\right)$$

for all $i \in \mathcal{C}, j \in \mathcal{V}$, where $\mathbf{f}_\mathcal{C}$, $\mathbf{f}_\mathcal{V}$, $\mathbf{g}_\mathcal{C}$ and $\mathbf{g}_\mathcal{V}$ are 2-layer perceptrons with *relu* activation functions [1]. The policy is obtained by discarding the constraint nodes and applying a final 2-layer perceptron on variable nodes with a softmax activation function to produce a probability distribution over the candidate branching variables. To initialize the weights and stabilize the learning procedure, the authors adopt a simple affine transformation $\mathbf{x} \leftarrow (\mathbf{x} - \beta)/\sigma$, which the authors call a prenorm layer, applied after the summation in the convolution layer.

## 4 Experimental methodology

Within the experiments described in the paper, we focus on replicating the GCNN results for the combinatorial auction problem. As a baseline, we reproduce the results of the reliability pseudocost (RPB) method, a variant of hybrid branching [12], which is used by default in SCIP. Unfortunately, not every experiment or baseline result was reproduced due to lack of time and computational constraints. However, we were able to extend the authors' ablation study to the combinatorial auction problem.

The purpose of extending the authors' ablation study is to to validate the model's architecture. In the original paper, the authors present an ablation study of their proposed GCNN model on the set-covering problem by comparing three variants of their convolution operation: mean convolution

(MEAN), sum convolution without the prenorm layer (SUM), and the sum convolution with prenorm layer (GCNN). We reproduce these variants of the GCNN model using the combinatorial auction problem.

We found that the paper was well written and very amenable to reproduction. The authors provided their code, which we used to reproduce the experiments. The supplementary material attached to the paper provided sufficient details to replicate the paper in its entirely, from collecting the training dataset to running the experiments.

### 4.1 Dataset collection

The authors not only provide the code to obtain the MILP training dataset, but also provide the code they used to generate the MILP problem instances, enabling us to easily collect the training, validation, and test datasets. First, we generate the MILP problem instances. For the combinatorial auction problem, we train and test on instances with 100 items for 500 bids, and we evaluate on instances with 100 items and 500 bids (Easy), 200 items for 1,000 bids (Medium) and 300 items for 1,500 bids (Hard). In total, we generate 10,000 random instances for training, 2,000 random instances for validation, and 3x20 instances for testing (20 easy instances, 20 medium instances, and 20 hard instances). Then, we use SCIP's strong branching rule to solve the combinatorial auction MILP instances from the corresponding training or validation sets, each time with a new random seed, and record the strong branching decision ($\mathbf{a}$) and extract the state representation ($\mathbf{s}$) at each node. This yields datasets of state-action pairs $\mathcal{D} = \{(\mathbf{s}, \mathbf{a})\}$. Like the authors, we continued to process new instances by sampling with replacement, until we achieved 100,000 samples for training, and 20,000 for validation. The computational challenges related to the dataset collection are detailed in the section on reproducibility costs.

### 4.2 Implementation details

The authors' repository includes detailed instructions on how to install and set up all the required libraries, and the supplementary section to their paper clearly describes the machine used as well as the model's architecture. We executed our code in a Jupyter Notebook on Google Cloud Platform (GCP) in a server with 8 cores, 30G of RAM, and an Nvidia Tesla V100 GPU. To collect the training samples, we used a GCP server with 34 cores and 190G of RAM. Throughout all experiments, we used SCIP 6.0.1 as the backend solver with a solving time limit of 1 hour.

We decided to implement the architecture described in the paper using the authors' code. Like the authors, we trained the GCNN model using the Tensorflow library. First, the parameters of the prenorm layers are initialized with the empirical mean and standard deviation of x on the training dataset, and fixed before the actual training begins. Then, the cross-entropy loss is minimized using the Adam optimizer [11] with minibatches of size of 32 and an initial learning rate of 0.001. We divide the training loss by 5 when the validation loss does not improve for 10 epochs, and stop training if it does not improve for 20 epochs. On average training the GCNN on the combinatorial auction problem took 150 epochs per seed. Different sets of hyperparameters were not explored due to lack of time. Hyperparameter search can be quite difficult for these benchmark problems due to the prohibitive time it takes to solve MILP problems. We elaborate on the computational costs in the following subsections.

As part of the reproducibility challenge, we extend the authors' ablation study to the combinatorial auction problem. The ablation study is used to validate the authors' chosen architecture. As the authors opt for un-normalized convolutions, a prenorm layer is added to stabilize the learning procedure and prevent weight initialization issues. We run the three proposed configurations: sum convolutions without a prenorm layer, mean convolutions and the sum convolutions with a prenorm layer.

To evaluate the results, we run the GCNN model, the RPB baseline, and the models from the ablation study on 20 new instances for each problem difficulty (Easy, Medium, Hard) using 5 different seeds, which amounts to 100 solving attempts per method. These results are included in Section 4.

### 4.3 Reproducibility cost

The following section will describe the cost of reproducing the paper's results in terms of resources. The authors describe some of the computational resources required to run the experiments in the paper, and discuss training and inference time in detail. As we can see in table 1, which was obtained from the supplementary material of the original paper, training time per seed is lengthy. Based on this table, the total estimated time needed to train the baselines for all NP-Hard problems is of 68.85 hours. The estimated time needed to run the GCNN for all experiments is of 204.5 hours. In our case, training the GCNN on the combinatorial auction samples took 4 hours per seed with 5 seeds for a total of around 20 hours using an Nvidia Tesla V100 GPU. The training time per seed was similar for the ablation study.

Generating the training samples was also computationally expensive. Since we have to run the strong branching rule to collect training samples, the process is not trivial. In the original paper, the strong branching rule is referred to as a "slow expert" by the authors with reason. Since each benchmark represents an NP-Hard problem, running strong branching at every node is prohibitive. For example, the paper shows that solving time for the hard instances of the set-covering problem surpassed the 1 hour time limit set by the authors. Indeed, generating the samples for the set-covering problem, combinatorial auction problem, and facility location problem using 24 cores took 24 hours, 12 hours, and 20 hours respectively. In the case of the set-covering and facility location problems, our 190G RAM server ran out of memory before the validation samples finished generating. This was one of the limitations which led us to replicate the combinatorial auction problem. For future reproductions, we recommend generating each dataset separately to preserve memory.

Overall, we deem the computational cost of running the experiments to be high, which can hinder the reproduction of the results. Based on the authors' notes, we estimate it would take over 500 hours to reproduce all the experiments described in the paper, with around 150 hours needed to generate the datasets, 287 hours to train and evaluate the baselines, and 230 hours to train and evaluate the GCNN. In our case, it took around 110 hours to generate the data, and to train and evaluate all the methods we explore in our reproduction. To facilitate future reproduction efforts, a set of instances and corresponding training dataset could be provided with the code. Additionally, although we maintained frequent communication with the authors, all necessary implementation details for reproduction are included in the paper and the authors' code.

Table 1: Training time for each machine learning method from the original paper, in hours.

| Model | Set Covering | Combinatorial Auction | Capacitated Facility Location | Maximum Independent Set |
|---|---|---|---|---|
| TREES | $0.05 \pm 0.00$ | $0.03 \pm 0.00$ | $0.16 \pm 0.01$ | $0.04 \pm 0.00$ |
| SVMRANK | $1.21 \pm 0.01$ | $0.17 \pm 0.06$ | $1.04 \pm 0.03$ | $1.19 \pm 0.02$ |
| LMART | $2.87 \pm 0.23$ | $2.47 \pm 0.26$ | $1.38 \pm 0.15$ | $2.16 \pm 0.53$ |
| GCNN | $14.45 \pm 1.56$ | $3.84 \pm 0.33$ | $18.18 \pm 2.98$ | $4.73 \pm 0.85$ |

## 5 Results

The results of our comparative experiments are summarized in tables 2 and 3 and include some of the results from the original paper for comparison. The results of our reproduction are denoted by [RE], while the authors' results are denoted by [OR]. In table 4 and 5, we display the results of our ablation study.

### 5.1 Comparative experiment

In terms of prediction accuracy, GCNN performed comparatively to the original paper for the combinatorial auction problem. We did not reproduce the exact results due to the random nature of the generated instances. GCNN outperforms SCIP's default branching rule RPB in terms of running time and node count for every configuration. Furthermore, as in the original paper, we see that the GCNN generalizes well to instances of larger size than seen during training. Although the model was trained on combinatorial auction instances with 100 items and 500 bids (Easy), it was able to generalize to larger instances with 200 items and 1000 bids (Medium) and 300 items and 1500 bids (Hard). Thus, our findings confirm those found in the original paper. This was a particularly interesting result to validate as it indicates that a GCNN model could be used to improve current solvers by speeding up mixed-integer linear programming.

Table 2: Imitation learning accuracy on the test sets.

| Model | acc@1 | acc@5 | acc@10 |
|---|---|---|---|
| [RE] GCNN | 60.6 $\pm$ 0.1 | 90.7 $\pm$ 0.1 | 97.7 $\pm$ 0.1 |
| [OR] GCNN | 60.8 $\pm$ 0.2 | 90.8 $\pm$ 0.1 | 97.6 $\pm$ 0.0 |

Table 3: Policy evaluation on separate instances in terms of solving time, number of wins (fastest method) over the number of solved instances, and number of resulting branch-and-bound nodes (lower is better).

| | Easy | | | Medium | | | Hard | | |
|---|---|---|---|---|---|---|---|---|---|
| Model | Time | Wins | Nodes | Time | Wins | Nodes | Time | Wins | Nodes |
| [RE] GCNN | 2.39 $\pm$ 5.2 | 97/100 | 69 $\pm$ 11.6 | 13.56 $\pm$ 7.1 | 100/100 | 659 $\pm$ 13.5 | 154.9 $\pm$ 5.7 | 87/100 | 8028 $\pm$ 11.1 |
| [RE] RPB | 3.52 $\pm$ 8.1 | 3/100 | 11 $\pm$ 31.7 | 23.13 $\pm$ 7.2 | 0/100 | 696 $\pm$ 20.7 | 188.8 $\pm$ 7.7 | 13/100 | 9128 $\pm$ 11.1 |
| [OR] GCNN | 1.85 $\pm$ 5.0 | 100/100 | 70 $\pm$ 12.0 | 10.29 $\pm$ 7.1 | 100/100 | 657 $\pm$ 12.2 | 114.16 $\pm$ 10.3 | 87/100 | 5169 $\pm$ 14.9 |
| [OR] RPB | 2.74 $\pm$ 8.1 | 0/100 | 10 $\pm$ 32.1 | 17.41 $\pm$ 6.6 | 0/100 | 689 $\pm$ 21.2 | 136.17 $\pm$ 7.9 | 13/100 | 5511 $\pm$ 11.7 |

## 5.2 Ablation study

In the original paper, the authors present an ablation study of their proposed GCNN model on the set-covering problem by comparing three variants of their convolution operation: mean convolution (MEAN), sum convolution without the prenorm layer (SUM), and the sum convolution with prenorm layer (GCNN) used in the comparative experiments. We extended this ablation study to the combinatorial auction problem. The authors found that the solving performance of both MEAN and SUM variants was similar to that of the baseline GCNN for small instances, but performed significantly worse in terms of both solving time and number of nodes on larger instances. Our results partly reflect this trend as the solving performance of the variants MEAN and SUM is worse for the medium instances, but is similar to that of the baseline GCNN for the hard instances. It is worth noting, however, that some hard instances solved using the MEAN and SUM variants failed to run within the 1 hour solving limit, which could potentially skew the results. Therefore, it does seem like the sum-convolution offers a better architectural prior than mean-convolution. However, our results do not help confirm the authors' hypothesis that the prenorm layer helps stabilize training [1].

Table 4: Ablation study accuracy on the test sets.

| Model | acc@1 | acc@5 | acc@10 |
|---|---|---|---|
| GCNN | **60.6** $\pm$ **0.1** | 90.7 $\pm$ 0.1 | 97.7 $\pm$ 0.1 |
| MEAN | 60.7 $\pm$ 0.6 | **90.4** $\pm$ **0.2** | 97.5 $\pm$ 0.1 |
| SUM | 60.7 $\pm$ 0.3 | **90.4** $\pm$ **0.2** | **97.5** $\pm$ **0.0** |

Table 5: Ablation study policy evaluation on separate instances in terms of solving time, number of wins (fastest method) over the number of solved instances, and number of resulting branch-and-bound nodes (lower is better).

| | Easy | | | Medium | | | Hard | | |
|---|---|---|---|---|---|---|---|---|---|
| Model | Time | Wins | Nodes | Time | Wins | Nodes | Time | Wins | Nodes |
| GCNN | 2.84 $\pm$ 5.8 | **43/100** | **69** $\pm$ **11.6** | **16.61** $\pm$ **7.2** | **40/100** | **659** $\pm$ **13.5** | 154.9 $\pm$ 5.7 | 7/100 | 8028 $\pm$ 11.1 |
| MEAN | 2.93 $\pm$ 5.7 | 15/100 | 71 $\pm$ 13.2 | 17.16 $\pm$ 7.9 | 24/100 | 673 $\pm$ 14.8 | **129.46** $\pm$ **7.6** | 36/90 | 7011 $\pm$ 10.1 |
| SUM | **2.83** $\pm$ **6.0** | 42/100 | 71 $\pm$ 12.1 | 17.68 $\pm$ 17.0 | 36/100 | 705 $\pm$ 22.8 | 144.58 $\pm$ 45.1 | **50/83** | **6998** $\pm$ **23.6** |

# 6 Discussion of findings

We were able to validate the authors' hypothesis that the GCNN models can improve the solving time of combinatorial optimization problems for the NP-Hard combinatorial auction benchmark and that

GCNN models trained on small instances can generate to larger instances. We also extended the ablation study performed by the authors to the combinatorial auction benchmark and our results showed that the difference in performance between different architecture choices were not as pronounced for the combinatorial auction problem compared to the set-covering problem.

Overall, the paper was clearly written, implementation details were well documented, and the authors provided their code making it easy to replicate the results reported. However, we still encountered some problems during the replication.

The process of generating the 120,000 samples (state-action pairs) was the most time consuming aspect of the reproduction. The data generation process took from 12 to 35 hours even when using multiple CPUs. The resulting datasets were of 4GB which made them difficult to work with. The time it took to generate training instances speaks to the importance of this paper in addressing solver efficiency. An additional problem we encountered was with the compatibility of the libraries and the proposed environment. For instance, the version of full strong branching, *vanillafullstrong*, developed by the authors was removed from subsequent versions of PySCIPOpt, meaning the package was no longer up to date. A similar issue arose with the implementation of Tensorflow's *contrib* library, which is no longer supported in recent Tensorflow releases. These compatibility issues may become a barrier to future reproducibility efforts, and solutions for version control should be explored.

## 7 Conclusion

In this reproduction challenge, we were able to generate the sample datasets, train the GCNN model using behavioral cloning, and obtain results demonstrating that GCNN improves the policy for selecting branching variables. We test the resulting policies on problem instances of different sizes, and conclude that it is indeed possible to train on smaller instances and obtain a policy that improves the performance of the solver on larger instances. We also extended the authors' ablation study to the combinatorial auction problem, and confirmed the conclusion that using sum-convolution improves accuracy. As a result, we have shown that the paper is reproducible and have helped corroborate some of the authors' results.

In terms of reproducibility, we would like to commend the authors on their ability to convey such a complex topic in an accessible manner. We found the explanations to be clear and sufficiently detailed to facilitate reproduction. Furthermore, we thank the authors for being willing to answer our questions throughout the challenge.

To conclude, we offer the following suggestions to improve the reproducibility of the paper:

- Publish a dataset of training and validation samples to decrease the overall time involved in reproducing the results.
- Update the code to work with the newer versions of Tensorflow.

In future works, it would be interesting to extend the ablation study to the remaining NP-Hard problems to determine whether we can find further empirical evidence to support the architecture choices. It would also be interesting to conduct a hyperparameter search to determine whether the baseline performance can be improved.

## References

[1] Gasse, M., Chételat , D., Ferroni, N., Charlin, L. and Lodi, A. (2019). Exact Combinatorial Optimization with Graph Convolutional Neural Networks. *arXiv:1906.01629.*

[2] Wolsey, L.A. (1988). Integer Programming. *Wiley-Blackwell.*

[3] Land. H. A and Doig, A.G (1960). An automatic method of solving discrete programming problems *Econometrica* 28:497–520.

[4] Lodi, A. and Zarpellon G. (2017). On learning and branching: a survey. *TOP* 25:207–236.

[5] Khalil, B.E. Le Bodic, P. Song, L. Nemhauser, G. and Dilkina, B. (2016). Learning to branch in mixed integer programming. *In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* pages 724–731.

[6] Alvarez, A.M. Louveaux, Q. and Wehenkel, L. (2017). A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29:185–195.

[7] Hansknecht, C.  Joormann, I.  and Stiller, S. (2018). Cuts, primal heuristics, and learning to branch for the time-dependent traveling salesman problem. *arXiv:1805.01415*

[8] He, H. Daumé, H. III  and Eisner, J.(2014). Learning to search in branch-and-bound algorithms. *In Advances in Neural Information Processing Systems 27*, pages 3293–3301.

[9] Ronald, A.H. (1960). Dynamic Programming and Markov Processes. *MIT Press, Cambridge, MA.*

[10] Pomerleau, D.A. (1991). Efficient training of artificial neural networks for autonomous navigation.*Neural Computation,* 3:88–97.

[11] Kingma, D.P  and Ba, J. (2015). Adam: A method for stochastic optimization. *In Proceedings of the Third International Conference on Learning Representations.*

[12] Achterberg, T. and Berthold, T. (2009). Hybrid branching. *In Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 5547.

[13] Khalil, B.E., Dai, H., Zhang, Y., Dilkina, B. and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *In Advances in Neural Information Processing Systems* 30, pages 6348–6358.