[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 paper 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 conclude that the paper is reproducible. The code repository can be accesses through the following link: `https://github.com/audreyanneguindon/NeurIPS_2019`

1 Introduction

The paper [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 computational time. In order to achieve this, the authors train their model using behavioural cloning and an expert strong branching rule to produce smaller trees.

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, mainly that 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 related to the density of the graph. These properties allow the authors to encode the state of an MILP branch-and-bound even with its variable structure and size.

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

To determine the reproducibility of the paper and confirm its conclusions, the main questions we will answer are:

- Can we reproduce the reported results with the given code?
- Does a model trained on smaller problems generalize to larger instances?
- How can the code be further expanded to facilitate reproduction?

Preprint. Under review.

2 Related Work

Most NP-Hard computer problems (non-deterministic polynomial-time hardness) are examples of combinatorial 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].

Although branch-and-bound has been extensively studied, there is still a lack of deep mathematical understanding on the decision process involved in selecting among candidate variable to branch on [4]. The current literature has focused on computational studies of said process and has recently been combined with machine learning [5, 7, 6]. 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 on its applications. The authors' aim is to build a model that can be applied regardless of the problem formulation and that is able to generalize to instances larger than those used to trained the model. 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 [9, 8], 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. By using GCNN, the researchers are able to leverage the natural bipartite graph representation of MILP problems to avoid manual feature engineering [1], which one of the concerns the paper is trying 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 small search trees [5].

Previous work [5, 6, 7] has used imitation learning to learn a substitute function to replicate the strong branching strategy at reduced computational time. Following these imitation learning studies [10], the proposed model is trained by using behavioral cloning on the strong branching strategy. The selected policy is selected by minimizing the cross-entropy loss [1].

However, though the authors use imitation learning in their studies, they differ from previous work by contrasting their results against both state-of-the-art branching rules and related machine learning branchers [6, 5, 7]. They also evaluate their work on different complexities, proving that their method is able to generalize not only on larger instances but also on various types of NP-hard problems.

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

3 Methodology

3.1 Experiments

In the paper, the authors compare their GCNN model against three competing machine learning approaches and SCIP's default branching rule. The authors also perform an ablation study to validate their architectural choices. They evaluate their approach on four NP-hard problem benchmarks, namely, a set-covering, combinatorial auction, facility location, and maximum independent set instances problem.

Within the experiments described in the paper, we focus on replicating the GCNN results for the combinatorial auction problem and extending the ablation study to this problem. We also reproduced the results of the reliability pseudocost (RPB), a variant of hybrid branching [12] which is used by default in SCIP to form the baseline for our evaluation. Unfortunately, not every result from the paper was reproduced due to lack of time.

We found that the paper was well written and very amenable to reproduction. The authors provided their code, which included all details related to the implementation as well as the method to generate the training samples. The supplementary material attached to the paper provided sufficient details to replicate all experiments and also described the machine used.

3.2 Implementation details

The author's repository includes detailed instructions on how to install and set up all the required libraries, and the supplementary section to their paper describes the machine used as well as the model's architecture. We executed our code in Jupyter Notebook on Google Cloud Platform (GCP) in a server with 8 cores, 30G of RAM, and 2 Nvidia Tesla V100 GPUs. To generate the training samples, we used a GCP server with 34 cores and 190G of RAM. Training the models was done with Tensorflow. Throughout all experiments, we used SCIP 6.0.1 as the backend solver with a time limit of 1 hour.

In order to replicate the paper's baseline, we maintained the GCNN architecture described by the authors. First, each prenorm layer is pretrained sequentially on the training dataset. Then, the authors minimize the cross-entropy loss using the Adam optimizer [11] to learn the branching policy with a batch size of 32 and an initial learning rate of 0.001. They 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. These parameters were untouched in our implementation 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 our reprodubility challenge, we run the presented ablation study. 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, mean convolutions and the sum convolutions with a prenorm layer. This results are included in Section 4.

3.3 Training samples

The authors provided the code which allowed us to easily collect the training, validation and test samples from the instances. The process records strong branching decisions and extract bipartite state representations during branch-and-bound on a collection of instances. This yields a dataset of state-action pairs. Like the authors, we generate 100,000 branching samples extracted from 10,000 randomly generated instances for training, 20,000 branching samples from 2,000 instances for validation, and the same for testing. Since we have to run the strong branching rule to collect training samples, the process was 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. Indeed, generating the samples for the set-covering problem, combinatorial auction problem, and facility location problem took 24 hours, 12 hours, and 20 hours on 8 cores respectively. In the case of the set-covering and facility location problems, our 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 the training samples and the validation and testing samples separately to preserve memory.

3.4 Reproducibility Cost

The authors describe some of the computational resources required in their paper, but this is not discussed in detail. Training the GCNN on the combinatorial auction samples took 4 hours per seed with 5 seeds for a total of around 20 hours using the Tesla V100 GPU. The authors shared with us that average training time per seed was about 14 hours (a total of 70 hours) for GCNN on the set-cover problem, which also ran on a Tesla V100 GPU. As previously mentioned, generating the training samples was also computationally expensive as it involved strong branching. In fact, combinatorial optimization problems inherently have a polynomial time complexity. For example, the paper shows that the hard instances of the set-covering problem in the best case had a solving time of 28 minutes. Overall, we deem that the computational cost of running the experiments is high, which can hinder the reproduction of the results. To facilitate future reproduction efforts, a set of instances and corresponding training samples could be provided with the code.

4 Results

The results of our comparative experiments are summarized in tables 1 to 3 and include some of the results from the original paper for comparison. In table 4, we display the results of our ablation study.

4.1 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). 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 programming.

Table 1: 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 2: Policy evaluation on separate instances in terms of number of wins (fastest method) over the number of solved instances, and number of resulting branch-and-bound nodes (lower is better).

		Wins	Nodes	Reported Wins	Reported Nodes
Easy	GCNN	97/100	69 $\pm$11.6	25/100	70 $\pm$12.0
	RPB	3/100	11 $\pm$31.7	0/100	10 $\pm$32.1
Medium	GCNN	100/100	659 $\pm$13.5	99/100	689 $\pm$21.2
	RPB	0/100	696 $\pm$20.7	0/100	657 $\pm$12.2
Hard	GCNN	92/100	8028 $\pm$7.5	87/100	5169 $\pm$14.9
	RPB	8/100	9128 $\pm$11.1	13/100	5511 $\pm$11.7

Table 3: Policy evaluation on separate instances in terms of solving time.

		Easy	Medium	Hard
Reproduction	GCNN	2.39 $\pm 5, 2$	13.56 $\pm$7.1	154.9 $\pm$5.7
	RPB	3.52 $\pm$8.1	23.13 $\pm$7.2	188.80 $\pm$7.7
Original	GCNN	1.85 $\pm$5.0	10.29 $\pm$7.1	114.16 $\pm$10.3
	RPB	2.74 $\pm$7.8	17.41 $\pm$6.6	136.17 $\pm$7.9

4.2 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 problem of combinatorial auction. The authors found that the the solving performance of both variants MEAN and SUM was similar to that of the baseline GCNN on small instances, but performed significantly worse in terms of both solving time and number of nodes on harder instances. Our results show a similar trend, with a narrower gap for small and medium instances. These results help confirm the authors' hypothesis that sum-convolutions offer a better architectural prior than mean convolution and that the prenorm layer helps stabilize training [1].

Table 4: Ablation results in terms of solving time, number of wins and number of resulting nodes.

		Time	Wins	Nodes
Small				
	GCNN	2.84 $\pm$5.8	43/100	69 $\pm$11.6
	MEAN	2.93 $\pm$5.7	15/100	71 $\pm$13.2
	SUM	2.83 $\pm$6.0	42/100	71 $\pm$12.1
Medium				
	GCNN	16.61 $\pm$7.2	40/100	659 $\pm$13.5
	MEAN	17.16 $\pm$7.9	24/100	673 $\pm$14.8
	SUM	17.68 $\pm$17.0	36/100	705 $\pm$22.8

Table 5: Ablation results in terms of accuracy on the test sets.

	Model	Accuracy	
Small	GCNN	60.6 $\pm$	0.1
	MEAN	60.7 $\pm$	0.6
	SUM	60.7 $\pm$	0.3
Medium	GCNN	90.7 $\pm$	0.1
	MEAN	90.4 $\pm$	0.2
	SUM	90.4 $\pm$	0.2
Hard	GCNN	97.7 $\pm$	0.1
	MEAN	97.5 $\pm$	0.1
	SUM	97.5 $\pm$	0.0

5 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

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 reproducibility. 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. Furthermore, 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, which is no longer supported.

6 Conclusions

We have shown that the paper is reproducible and that it is possible to achieve the presented results. Though we faced some minor drawbacks in the implementation, the authors were willing to answer our questions and guide us.

We would like to comment briefly on the writing of the paper. As students in a machine learning, we found the explanations to the point and commend the authors in their ability to clearly convey such complex information. The reasoning was easy to follow and the authors were able to communicate the need for this research and its benefits in practice.

We would offer the following suggestions to improve the reproducibility of the paper:

- Publish the training and test instances as well as the training samples to decrease the overall time complexity involved in reproducing the results.
- Update the code to work with newer versions of Tensorflow.

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. 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. 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. Advances in Neural Information Processing Systems 27, pages 3293–3301

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

[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*