Strong Regularization is All You Need : Toward ML4CO Challenge

Seungjoon Park *
KAIST
sjoon.park@kaist.ac.kr

Minu Kim *
KAIST
minu.kim@kaist.ac.kr

Minchan Jeong
KAIST
mcjeong@kaist.ac.kr

Yejin Cho
Korea University
bethesunshine@korea.ac.kr

Mingyu Kim
KAIST
callingu@kaist.ac.kr

Abstract

Mixed Integer Linear Programming(MILP) is one of the representative branches of non-convex combinatorial optimization problems, and includes various problem types such as Traveling Salesman Problem(TSP) and Bin Packing Problem(BPP). The branch-and-bound algorithm is a tree search-based algorithm for solving such MILP problems, that gradually reduces the solution space by dividing a given MILP problem into sub-MILPs. It is known that the branching variable selected at each node of the branch-and-bound tree has an important effect on the performance of the algorithm. In this study, we aim to approximate effectively a strong branching policy, an expert strategy empirically known to perform best a priori, via an imitation learning model. We expressed each MILP problem as a featured bipartite-graph and imitation-learned the expert policy through GCNN. We confirmed that applying strong weight decay in the learning process can significantly decrease the time-dual integral bound.

1 Introduction

Among combinatorial optimzation problems, Mixed Integer Linear Programming(MILP) is a widely adopted class of problem in many of the real-world domains such as industrial engineering, economics and business. MILPs are known to be NP-complete. Hence, tree-search based algorithms are mainly used to find the optimal solution of MILP instances. The branch-and-bound(B&B) algorithm is one of such algorithms. It iteratively reduces the solution space in a search-tree (branch) to close the primal-dual gap (bound).

It is well known that the choice of the branching variable selected at each node of the B&B tree has a significant effect on the overall performance of the algorithm.[1] However, all existing branching strategies were compared and analyzed based on a priori experimental results. For instance, strong branching strategy – which greedily chooses a locally best branching variable after solving all sub-MILPs of each child node – is generally known to make the smallest B&B search tree, but requires an extensive amount of computation before each branching step. In case of pseudocost branching strategy, it requires a relatively few number of operations within each branching step, but its empirical quality of the branching underperforms that of the strong branching strategy.[1] In general, the optimal branching strategy of the B&B is yet to be known. In fact, optimal branching variable selection is NP-hard in the worst case, even at the root node of the B&B tree.[2]

*Equal contribution

NeurIPS 2021 Competition - Machine Learning for Combinatorial Optimization (ML4CO).

Many of the recent studies attempt to learn existing branching policy using machine learning techniques to reduce the operation costs and search for better branching policies. A stream of work leverage graph convolutional neural network (GCNN) and imitation learning strategy for a variable selection in B&B [3, 4, 5]. Encoding branching policies into a GCNN reduces the amount of manual feature engineering. In continuation with these efforts, NeurIPS 2021 Competition: Machine Learning for Combinatorial Optimization(ML4CO) aims to replace the heuristic components in traditional combinatorial problem solvers with machine learning techniques that achieves the lowest dual integral over time. (Dual Task)

In this work, we propose a winning model of the NeurIPS 2021 ML4CO Competition. Our model is primarily built upon the structure proposed by [3], a combinatorial optimizer based on GCNN with behavior cloning. While fitting the model on three competition problem benchmarks, we addressed several difficulties such as over-fitting and computation problem related to large instance. Our team made the following contribution to arrive at a winning solution.

- **Strong regularization**

  The strong regularization encourages the policy to explore various action candidates. It aims to filter a prospective action set rather than picking the most probable one. It suggests solely using experts such as *strong branching* may not be enough.

- **Generalization of the learned policies**

  Our team observed that the model trained from a problem benchmark could be used for another benchmark when they have similar training and test constraint structures. We illustrate the applicability of our model trained on [item placement benchmark] to the [load balancing benchmark].

In section 2, we will introduce our detailed approach and resulting rewards on each problem benchmark (item placement, load balancing, and anonymous tasks). Further discussions and analyses will be covered in section 3.

2 Benchmarks

2.1 Evaluation Metric

We competed at the dual-task, of which the objective is to minimize the dual integral defined as follows. (See 1) In the definition of the dual-integral, T denotes time limit, $\mathrm{c}^{\mathrm{T}}\mathrm{x}^*$ denotes optimal solution and z_t^* denotes the best dual bound at time t which is depending on our learned policy. However, our tables do not include dual bound; instead, they include the average cumulative reward of the instances in the evaluation set. The cumulative reward is defined as $\int_{t=0}^{T} z_t^* dt$, the negated version of the dual-integral without $T\mathrm{c}^{\mathrm{T}}\mathrm{x}^*$. Because $T\mathrm{c}^{\mathrm{T}}\mathrm{x}^*$is an instance-specific constant that depends on the optimal solution, maximizing cumulative reward is equivalent to minimizing dual-integral.

$$T\mathrm{c}^{\mathrm{T}}\mathrm{x}^* - \int_{t=0}^{T} z_t^* dt \tag{1}$$

2.2 Evaluation Method

To evaluate our results, we used a single GeForce RTX 2080ti and RTX 1080ti. Moreover, we used our own evaluation pipeline, which is different from the official evaluation pipeline. To start with, we did not use the whole evaluation set. Instead, the evaluation was conducted using only 50 evaluation instances, of which a last index is an even number. In addition, because the official evaluation pipeline using 15 minutes as a time limit per instance is too long to test many candidate models, we used a 1-minute time limit for each instance. Secondly, there is inconsistency even in the average reward of the same model. As the evaluation metric depends on time, it is also dependent on the machine's processing speed. Because we tried and evaluated the various methods in different devices, we inevitably split the table according to the devices where we assessed the methods.

2.3 Experimental Details

2.3.1 Item Placement and Load Balancing Task

In Table 1, we compare three models, Baseline, Weight Decay, Finetuning. The baseline is the model proposed in [3] which uses the GCNN layer to integrate the knowledge of constraints, variables, and edges. The weight decay model has the same structure as the baseline model and is trained with the weight decay. The finetuning models are initialized using the pretrained weights of the weight decay models. After that, we froze up to the GCNN layers and trained only the classifier for eight epochs using the baseline setting to construct the finetuning model. The finetuning models outperform baseline models and weight decay models. In Addition, a standard deviation of the finetuning model is larger than that of the Baseline model. We observed that the finetuning model gets a higher reward than the others when the initial lower bound of the instance is high. Likewise, the finetuning model gets a lower reward than the others when the initial lower bound of the instance is low. In conclusion, we conjecture that strong regularization on the GCNN layer helps the agent work well in instances with a higher chance of high reward.

Table 1: Average reward of finetuning, weight decay and baseline models on the item placement task. **Bold entries** indicates the submitted model.

	Baseline	Weight Decay	Finetuning	
		($\lambda = 0.9$)	($\lambda = 0.9$)	($\lambda = 1.5$)
Avg. Reward	265.427	292.733	**330.182**	320.139
Max. Reward	899.452	948.370	1345.099	1395.967
Min. Reward	**21.9330**	0.0009	0.0009	0.0009
Std	194.647	258.326	301.590	304.903

Table 2: Average reward of combined models, weight decay and baseline models on the item placement task. **Bold entries** indicates the best results.

	Baseline	Weight Decay	Finetuning	Combined Models
All node features	287.244	-	362.599	**370.790**
Pseudo Cost	270.777	201.541	-	365.067

We investigate which features are crucial information for determining the reward. (See Table 2). The combined model uses either a baseline or a finetuning model depending on the initial lower bound of the instance. Using a kind of additional heuristic to select which model to be used, the combined model is the best among the models we tested. All node features use all given features to train the model. Pseudo Cost uses pseudo cost only as a feature.

Regarding the load balancing task, we could not train the model in our device. In the case of the load balancing task, a problem is too huge to handle in our device. (See Figure 1) Therefore we reuse the best finetuning model trained on the item placement task for the load balancing task without any fine-tuning.

2.3.2 Anonymous Task

We used a baseline model as provided in [3] for the anonymous task. The baseline model outperforms any of the simple models using pseudo cost observations provided in the Ecole library, and their finetuning variants. (See Table 3.) Unlike as in the Item Placement and Load Balancing tasks, neither the strong regularization nor the pre-trained transferred model from the Item Placement task worked better than the baseline model trained in this problem set. (See Table 4.) Further observations are to be dealt in section 3.

Table 3: Average reward of finetuning, weight decay and baseline models on the anonymous task **Bold entries** indicates the submitted model.

Baseline	Pseudocost		
	pseudo cost only	with scaled age + $\lambda = 0.1$	$\lambda = 1$
1.903	1.902	1.849	1.839

Reward scale = 10^6

Table 4: **Strong regularization does not properly perform in the Anonymous task**. Two baseline models are trained on the Anonymous benchmark data, with (**Baseline**) or without (**WD**) strong regularization, respectively. The last model (**Item Placement**) simply uses the trained weights from the Item Placement benchmark ($\lambda = 0.9$) as done in Load Balancing task. The average, maximum, and minimum rewards from 50 instances for each models are listed.

	Baseline	**WD** ($\lambda = 0.9$)	**Item Placement**
Avg. Reward	**1.341**	1.104	1.274
Max. Reward	7.160	**7.164**	7.159
Min. Reward	**0.111**	0.110	0.110

Reward scale = 10^6

3 Discussion

While regularization with the value of λ at around 5×10^{-4} is known to increase the models' generalization capability, little is known about the effect of the strong regularization. In this section, we attempt to briefly analyze its effect in different problem benchmarks.

Overall, strong regularization seems to improve the performance of the models when the data are sampled from problem-specific distributions. We have already observed that $\lambda \geq 0.9$ works well in Item Placement and Load Balancing tasks. However, this is not the case in the Anonymous task, where the problems are sampled from multiple distributions. (See Figure 1 and Table 4.)

Interestingly, the strong regularization *improves* the average reward, albeit *decreases significantly* the expert imitation accuracy. Figure 2 provides visualized data of the average reward and the imitation accuracy for different models in the Item Placement task. For problem distributions with high homogeneity, it seems that the expert imitation capability has little or no relation with the average reward. It is possible that strong regularization strengthens the effect of few important parameters of the model, whereas ignoring all other parameters to decrease the imitation capability. Such strategy may work well in general for some problem-specific data distributions with high homogeneity, but is expected to show less performance in heterogeneous data-independent problem distributions, as very few parameters are trained and utilized. This hypothesis coincides our observation as in Table 4 and Figure 2.

4 Conclusion

In this work, we have used strong regularization with $\lambda = 0.9$ on the baseline model of [3] to improve the performance in specific problem benchmarks. Its performance can be further improved by finetuning the final classifier layers in the baseline settings. While it is still open to question how strong regularization enhances the model performance in certain problem benchmarks, it is clear that the expert imitation accuracy does not significantly contribute to the model performance. This suggests the importance of exploring new branching policies that are yet to be known.

Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [No.2019–0–00075, Artificial

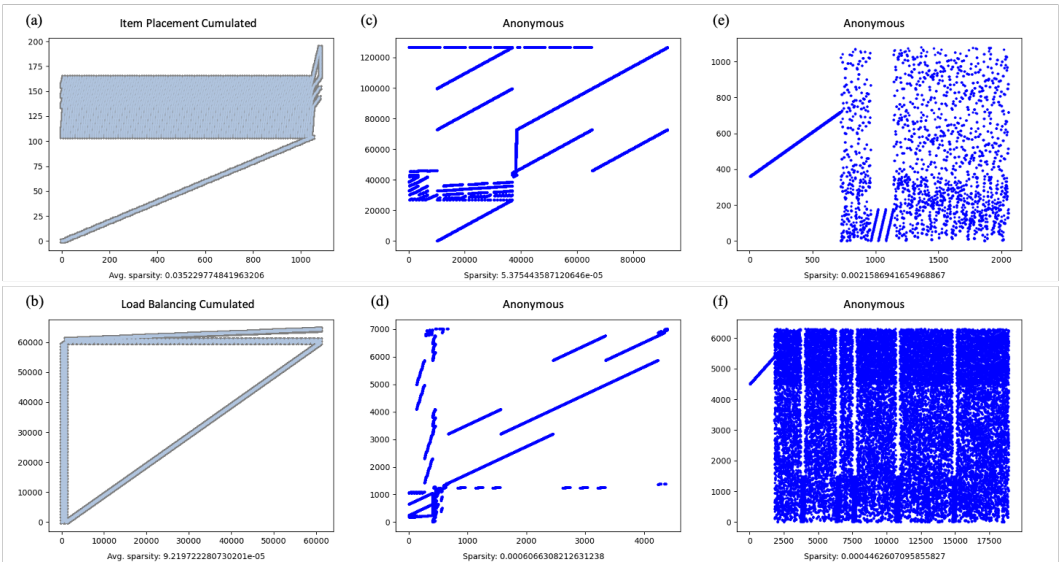

Figure 1: **Sparsity plot of constraints for different problem benchmarks**. Sparsity plots for 100 instances of constraint matrices of Item Placement task (a) and Load Balancing task (b) are overlapped with gray marks, together with a sparsity plot of a single instance via light blue marks. The constraint matrices for each instances are nearly identical within the two problem benchmarks. Four randomly chosen sparsity plots(training instances 85, 2, 41, 17 of Anonymous data) of the constraints in Anonymous task are given in (c)-(f). The Anonymous task includes a wider variety of problem from multiple heterogeneous distributions.

Intelligence Graduate School Program (KAIST)] and [No. 2021–0–00907, Development of Adaptive and Lightweight Edge-Collaborative Analysis Technology for Enabling Proactively Immediate Response and Rapid Learning].

References

[1] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] Paolo Liberatore. On the complexity of choosing the branching literal in dpll. *Artificial intelligence*, 116(1-2):315–326, 2000.

[3] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *NeurIPS*, 2019.

[4] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *NeurIPS*, 2018.

[5] Gil Lederman, Markus N. Rabe, Edward A. Lee, and Sanjit A. Seshia. Learning heuristics for automated reasoning through reinforcement learning. In *ICLR*, 2018.

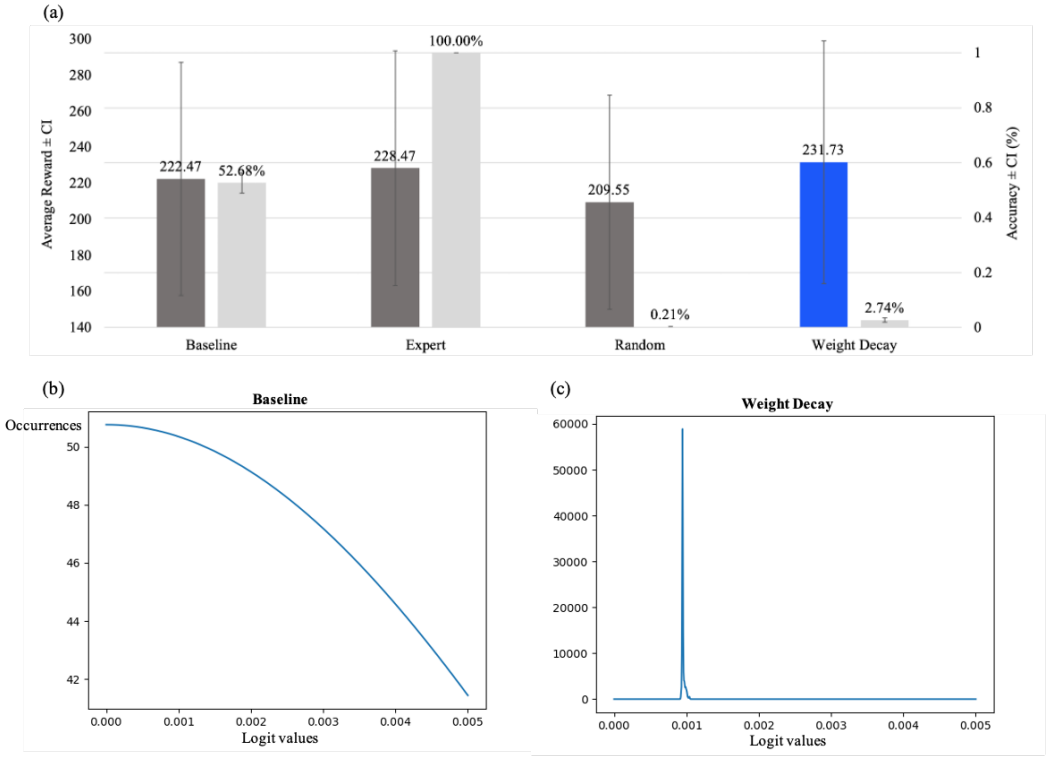

Figure 2: **Average rewards, imitation accuracy, and logit distribution of a single action set for Item Placement tasks in different models.** Average reward and the accuracy with respect to the expert for different models are given in (a). The model with highest average reward is depicted with a blue mark. Given below is the distribution of logit values (*i.e.* log-odds) of an action set (within a randomly chosen single step) for Baseline model (b) and Weight Decay ($\lambda = 0.9$) model (c). The logit values of Weight Decay model (c) are densely packed around the same value, resembling a random decision. However, the accuracy with respect to the expert is tenfold higher than that of the random policy.