A Bi-Level Framework for Learning to Solve Combinatorial Optimization on Graphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

Combinatorial Optimization (CO) has been a long-standing challenging research topic featured by its NP-hard nature. Traditionally such problems are approximately solved with heuristic algorithms which are usually fast but may sacrifice the solution quality. Currently, machine learning for combinatorial optimization (MLCO) has become a trending research topic, but most existing MLCO methods treat CO as a single-level optimization by directly learning the end-to-end solutions, which are hard to scale up and mostly limited by the capacity of ML models given the high complexity of CO. In this paper, we propose a hybrid approach to combine the best of the two worlds, in which a bi-level framework is developed with an upper-level learning method to optimize the graph (e.g. add, delete or modify edges in a graph), fused with a lower-level heuristic algorithm solving on the optimized graph. Such a bi-level approach simplifies the learning on the original hard CO and can effectively mitigate the demand for model capacity. The experiments and results on several popular CO problems like Directed Acyclic Graph scheduling, Graph Edit Distance and Hamiltonian Cycle Problem show its effectiveness over manually designed heuristics and single-level learning methods.

1 Introduction

Combinatorial Optimization (CO) is a family of long-standing optimization problems. A large portion of CO problems is NP-hard due to the combinatorial nature, raising challenges for traditional (exact) solvers on even medium-sized problems. Heuristic algorithms are often adopted to approximately solve CO problems within an acceptable time, and there is a growing trend adopting modern data-driven approaches to solve CO problems that achieve better and faster results [28].

The major line of works solving CO with machine learning (ML) is single-level [8, 24, 27, 28, 36, 38, 54, 56, 60], where the prediction of ML module lies in the solution space, assuming the model has enough capacity learning the input-output mapping of the CO problem. However, achieving such an assumption is non-trivial, leading to the following two aspects of challenges. On the one hand, it is challenging to design a model with enough capacity with limited computational resources, and existing models are usually tailored for specific problems which require heavy trail-and-error [24, 54, 56]. On the other hand, training such a heavy model requires either supervision from high-quality labels [29, 54, 57] which are infeasible to obtain for large-sized problems due to the NP-hard nature, or reinforcement learning (RL) [8, 28, 35, 36] which might be unstable due to the challenges of large action space and sparse reward especially for large-sized problems [49].

An alternative approach resorts to a hybrid machine learning and traditional optimization pipeline [11, 16, 29, 48, 57] hoping to utilize the power of traditional optimization methods. However, designing a general hybrid MLCO approach is still non-trivial, as existing methods [11, 57] usually require domain-specific knowledge for the model design. It is again challenging to obtain high-quality

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

supervision labels, and existing methods are based on either problem-specific surrogate labels [16, 29, 57], or learned with RL while the challenges of RL still exist [11, 48].

In this paper, we propose a general hybrid MLCO approach over graphs if all constraints are encoded by the graph. We first reduce the complexity of deep learning model by reformulating the original CO into a bi-level optimization, whose objective is to minimize the long-term upper-level objective by optimizing the graph structure, and the lower-level problem is handled by an existing heuristic. We resort to RL and the traditional heuristic can be absorbed as part of the environment, and it is shown that the sparse reward issue is mitigated for the resulting RL problem compared to previous RL-based methods [8, 28, 35, 36]. Specifically, our model is built with standard building blocks: the input graph is encoded by Graph Convolutional Network (GCN) [30], and the actor and critic modules are based on ResNet blocks [21] and attention models [52]. All modules are learned with the Proximal Policy Optimization (PPO) algorithm [45]. **The contributions of this paper include:**

- To combine the best of the two worlds, we take an initiative for developing a general hybrid approach that integrates traditional heuristic solvers with machine learning algorithm.
- We propose a bi-level optimization formulation for learning to solve CO on graphs. The upper-level optimization adopts a reinforcement learning agent to adaptively modify the graphs, while the lower-level optimization involves traditional learning-free heuristics to solve combinatorial optimization tasks on the modified graphs. Our approach does not require ground truth labels.
- The experiments for graphs up to thousands of nodes on several popular tasks such as Directed Acrylic Graph scheduling (DAG scheduling), Graph Edit Distance (GED), and Hamiltonian Cycle Problem (HCP) show that our method notably surpasses both traditional learning-free heuristics and single-level learning method. Our method generalizes well to graphs of different sizes while having comparable overhead to the single-level learning methods.

2 Related Work

Here we discuss related works listed in recent surveys [4, 58, 59]. We present their methodologies and compare with our method.

Learning the end-to-end solution as a sequence. The major line of existing MLCO works mainly focus on tackling the problem end-to-end by predicting a sequence of solutions [24, 27, 28, 36, 38, 54, 56, 60]. The pioneering work [54] is designated for traveling salesman problem (TSP), whereby the sequence-to-sequence Pointer Network (PtrNet) model is learned with supervised learning. In [28], a graph-to-sequence framework is proposed with graph embedding [10] and Q-learning [37] for general CO over graphs, and this general framework inspires the major line of MLCO works with applications to DAG scheduling [36], graph matching [33], and job shop scheduling [60]. The framework in [28] is also treated as the RL baseline in this paper. Our approach differs from these single-level end-to-end RL methods [28, 33, 35, 36], which lack the flexibility to borrow the power of traditional methods and often suffer from the fundamental issues in RL: sparse reward and large action space.

Learning to rewrite end-to-end solutions. Another line of end-to-end learning works predicts rewriting strategies of an existing solution [8, 35], where the model predictions also lie in the solution space. The agents learn to improve an existing solution in a decision sequence, and problems like job scheduling, expression simplification and routing problems have been studied. These learning-based local search heuristics also fall within the single-level paradigm, while our method incorporates another optimization over the graph structure. The sparse reward issue also exists for [8, 35], as a long episode is usually required for searching for a satisfying result.

One-shot unsupervised end-to-end learning. There is also a growing trend applying one-shot unsupervised methods for the input-output mapping of CO, with maximum clique and graph cut solved by [27], and quadratic assignment problem solved by [56]. However, the generalization ability of these methods is still an open question, and complicated constraints cannot be encoded in the network's prediction. Moreover, such a one-shot end-to-end network often calls for much higher model capacity compared with our multi-round alternating optimization paradigm.

Hybrid of machine learning and traditional solvers. Different from learning the solution end-to-end, researchers also propose hybrid machine learning and traditional solver approaches. ML modules are studied as sub-routines for traditional solvers, especially predicting branching strategies for branch-and-bound with either supervised learning [16, 29] or reinforcement learning [48]. In [57],

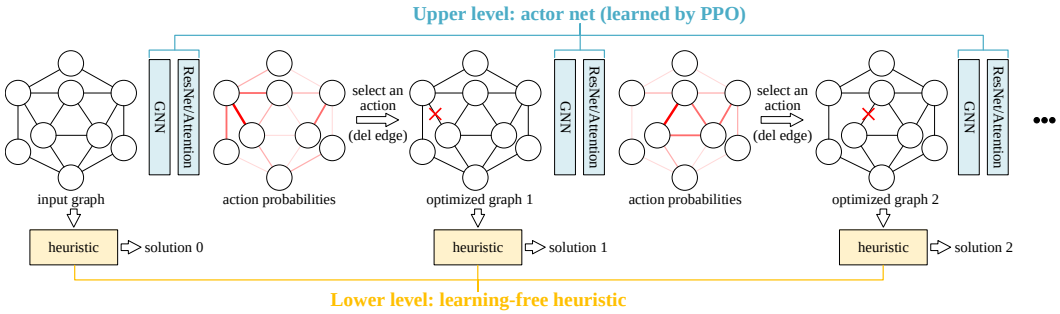


Figure 1: An overview of our bi-level hybrid MLCO solver. The graph structure is optimized at the upper level by an RL agent, and the optimized graphs are solved by heuristics at the lower level. The actions can be any modifications of the edges (i.e. adding, deleting edges and modifying edge attributes), and edge deletion is presented in this example.

the heuristic routine in A* algorithm is replaced by graph neural network to solve graph edit distance. However, these methods are tailored for special problems, and our aim is to develop a more general framework, where the learning part and heuristic module are two peers alternatively performed.

Bi-level optimization. Our method is based on bi-level optimization, which is a family of optimization problems where a lower-level optimization is nested inside an upper-level optimization. Bi-level optimization is in general NP-hard [25, 53], and the applications of bi-level optimization can be found ranging from multi-player games [25] to vision tasks [34], and there is a loosely relevant attempt [2] adopting supervised learning to solve a bi-level optimization for transportation.

3 Our Approach

In this paper, we propose a **Bi**-level **Hyb**rid (BiHyb) machine learning and traditional heuristic approach. Sec. 3.1 shows both single-level and bi-level formulations of CO, and Sec. 3.2 shows the RL approach to the bi-level CO.

3.1 Bi-level Reformulation of Combinatorial Optimization

Without loss of generality, we consider the classic single-level CO with a single graph $\mathcal{G}$ as:

$$\min_{\mathbf{x}} f(\mathbf{x}|\mathcal{G}) \qquad s.t. \quad h_i(\mathbf{x}, \mathcal{G}) \leq 0, \text{for } i = 1...I \tag{1}$$

where $\mathbf{x}$ denotes the decision variable (i.e. solution), $f(\mathbf{x}|\mathcal{G})$ denotes the objective function given input graph $\mathcal{G}$ and $h_i(\mathbf{x}, \mathcal{G}) \leq 0$ represents the set of constraints. For example, in DAG scheduling, the constraints enforce that the solution $\mathbf{x}$, i.e. the execution order of the DAG job nodes, lies in the feasible space and does not conflict the topological dependency structure of $\mathcal{G}$. The popular framework of existing MLCO methods regards Eq. 1 as a straight-forward end-to-end learning task, and various training methods have been developed, including: 1) supervised learning [38, 54] obtaining training labels by solving small-scaled Eq. 1 with traditional solvers, however it is nearly infeasible to solve larger NP-hard problems; 2) unsupervised learning [27, 56] adopting the continuous relaxation of Eq. 1 as the learning objective, but existing methods face challenges when dealing with complicated constraints; and 3) reinforcement learning [8, 28] by predicting $\mathbf{x}$ sequentially, but the reward signal is unavailable until $\mathbf{x}$ reaches a complete solution, leading to the sparse reward issue.

To ease the challenges introduced by the single-level formulation, we resort to the classic idea of modifying the original problem to aid problem solving, e.g. adding cutting planes for integer programming [17, 51]. Our proposed framework is capable of handling CO on graphs if all constraints can be encoded by the graph structure, and our motivation is described by the following proposition:

Proposition. *If all constraints are encoded by the graph structure, for $\mathcal{G}$ and its optimal solution $\mathbf{x}^*$, there must exist $\mathcal{G}^*$ modified from $\mathcal{G}$, such that $\mathbf{x}^*$ is the output of a heuristic algorithm by solving $\mathcal{G}^*$.*

Proof. The proof is relatively trivial by construction. Since all constraints are encoded by graph structure and $\mathbf{x}^*$ is a feasible solution with $\mathcal{G}$, we can add constraints to $\mathcal{G}$ until we reach $\mathcal{G}^*$ whose only feasible solution is $\mathbf{x}^*$. Thus, $\mathbf{x}^*$ is the output of any heuristic algorithm by solving $\mathcal{G}^*$. □

Algorithm 1: Policy Roll-out for Bi-level Learning of Hybrid MLCO Solver (BiHyb)

Input: Original graph $\mathcal{G}$; Max number of actions K.

$\mathcal{G}^0 \leftarrow \mathcal{G}$;

for $k \leftarrow 0 \ldots (K-1)$ **do**

  Predict $P(\mathbf{a}_1), P(\mathbf{a}_2|\mathbf{a}_1)$ on $\mathcal{G}^k$ and sample $\mathbf{a}_1, \mathbf{a}_2$; # upper-level optimization for learning

  $\mathcal{G}^{k+1} \leftarrow$ add, delete or modify the edge $(\mathbf{a}_1, \mathbf{a}_2)$ in $\mathcal{G}^k$; # state-transition

  $\mathbf{x}^{k+1} \leftarrow$ solve $\arg\min_{\mathbf{x}^{k+1}} f(\mathbf{x}^{k+1}|\mathcal{G}^{k+1})$ by heuristic algorithm; # lower-level optimization

  $r_k \leftarrow f(\mathbf{x}^k|\mathcal{G}) - f(\mathbf{x}^{k+1}|\mathcal{G})$; # reward

Output: A list of rewards $\{r_0 \ldots r_{K-1}\}$.

In practice, it may be difficult to find $\mathcal{G}^*$, but it suggests a promising direction via graph structure optimization. Motivated by this fact, we propose to improve the solving quality for heuristic algorithms by finding optimized (not necessarily optimal) graphs by learning based on the bi-level reformulation of the original single-level problem. We introduce the optimized graph $\mathcal{G}'$:

$$\begin{aligned} \min_{\mathbf{x}', \mathcal{G}'} \; f(\mathbf{x}'|\mathcal{G}) \qquad s.t. \quad & H_j(\mathcal{G}', \mathcal{G}) \leq 0, \text{for } j = 1 \ldots J \\ & \mathbf{x}' \in \arg\min_{\mathbf{x}'} \{ f(\mathbf{x}'|\mathcal{G}') : h_i(\mathbf{x}', \mathcal{G}') \leq 0, \text{for } i = 1 \ldots I \} \end{aligned} \tag{2}$$

where $f(\mathbf{x}'|\mathcal{G}), f(\mathbf{x}'|\mathcal{G}')$ are the objectives for upper- and lower-level problems, respectively. The lower-level problem is the CO given the optimized graph $\mathcal{G}'$, which is solved by a heuristic algorithm. The solved decision variable $\mathbf{x}'$ is further fed to the upper-level problem, whose objective $f(\mathbf{x}'|\mathcal{G})$ denotes the original CO objective computed by $\mathbf{x}'$. The upper-level constraints $H_j(\mathcal{G}', \mathcal{G}) \leq 0$ ensure that the feasible space of $\mathcal{G}'$ is a subset of $\mathcal{G}$, and $\mathcal{G}'$ has at most K modification steps from $\mathcal{G}$. The upper-level problem optimizes $\mathcal{G}'$ by an RL agent, by regarding Eq. 2 as the environment.

3.2 Reinforcement Learning Algorithm

We resort to reinforcement learning to optimize $\mathcal{G}'$ in Eq. 2, which can be viewed as a data-driven embodiment of the classic bi-level optimization method by solving two levels of problems alternatively [50]. In Sec. 3.2.1 we present the Markov Decision Process (MDP) formulation of the bi-level optimization in Eq. 2, in Sec. 3.2.2 we describe the PPO learning algorithm in our approach.

3.2.1 The MDP Formulation

Eq. 2 is treated as the learning objective and optimized by RL in a data-driven manner. In this section, we discuss the Markov Decision Process (MDP) formulation for adopting RL to this bi-level optimization problem. The policy roll-out steps are summarized in Alg. 1. In the following, $\mathcal{G}^0$ equals $\mathcal{G}$ meaning the original graph, and $\mathcal{G}^k (k \neq 0)$ equals $\mathcal{G}'$ meaning the modified graph after action k.

State. The current graph $\mathcal{G}^k$ is treated as the *state*, whose nodes and edges encode both the problem input and the current constraints. The starting *state* $\mathcal{G}^0$ represents the original CO problem.

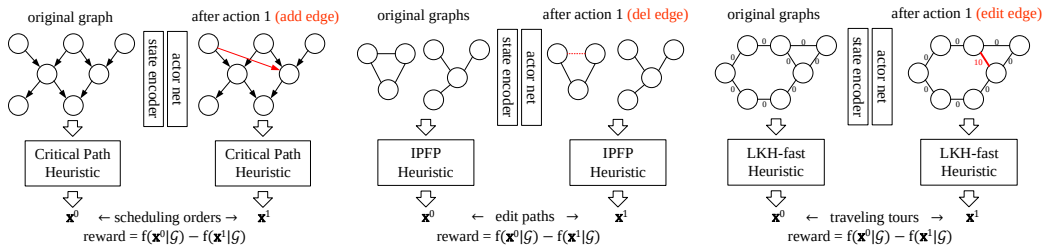
Action. The *action* is defined as adding, removing or modifying an edge in $\mathcal{G}^k$. Since there are at most m^2 edges if $\mathcal{G}^k$ has m nodes, we break the action space by decomposing the edge selection as two node selections: firstly selecting the starting node, and then selecting the ending node.

State transition. After taking an *action*, $\mathcal{G}^k$ transforms to $\mathcal{G}^{k+1}$ with one edge modified. The new graph $\mathcal{G}^{k+1}$ is regarded as the new *state* and is adopted for *reward* computation. The episode ends when it reaches the max number of actions K. In our implementation, we empirically set $K \leq 20$ for graphs up to thousands of nodes, therefore the sparse reward issue is mitigated compared to single-level RL methods (20 actions v.s. 1000+ actions per episode).

Reward. The new graph $\mathcal{G}^{k+1}$ results in a modified lower-level optimization problem whose objective becomes $f(\mathbf{x}^{k+1}|\mathcal{G}^{k+1})$, and $\mathbf{x}^{k+1}$ is solved by an existing heuristic. The *reward* is computed as the decrease of upper-level objective function given $\mathbf{x}^{k+1}$: reward $= f(\mathbf{x}^k|\mathcal{G}) - f(\mathbf{x}^{k+1}|\mathcal{G})$.

3.2.2 Proximal Policy Optimization (PPO)

We resort to the popular Proximal Policy Optimization (PPO) [45] as the RL framework. PPO is the simplified version of Trust Region Policy Optimization (TRPO) [44] where the model update is



(a) DAG Scheduling (b) Graph Edit Distance (c) Hamiltonian Cycle Problem

Figure 2: Illustration of the state, action and reward for all problems discussed in this paper.

restricted within a "trust region" to avoid model collapse. PPO is easier to implement than TRPO with comparative performance [45], maximizing the objective: $L(\theta) = \min(r_\theta \cdot A,\ clip(r_\theta, 1-\epsilon, 1+\epsilon) \cdot A)$ where r_θ is the importance sampling ratio parameterized by model parameter θ, A is the advantage value computed by the discounted accumulative reward minus the critic network prediction, and ϵ is a hyperparameter controlling the boundary of trust region. Some common policy-gradient training tricks are also adopted: we normalize the accumulated rewards during model update, and we add an entropy regularizer to encourage exploration beyond local optimums.

4 Experiments and Case Studies

We show the implementations and experiments on three challenging CO problems: DAG scheduling in Sec. 4.1, graph edit distance (GED) in Sec. 4.2, and Hamiltonian cycle problem (HCP) in Sec. 4.3. Our bi-level RL method PPO-BiHyb is compared with learning-free heuristics, and a single-level RL peer method namely PPO-Single following the general framework [28], which also covers the majority of RL-based methods [14, 24, 31, 33, 36, 60]. The model capacity and training/inference time of PPO-Single are kept in line with PPO-BiHyb for fair comparison.

4.1 Case 1: DAG Scheduling

The Directed Acyclic Graph (DAG) is the natural representation of real-world jobs with dependency, and the DAG scheduling problem is the abstraction of parallel job scheduling in computer clusters: each node represents a computation job with a running time and a resource requirement, and the node may have several parents and children representing data dependency. The cluster has limited total resources, and jobs can be executed in parallel if there are enough resources and the concurrent jobs do not have data dependency. Such scheduling problem is in general NP-complete [13], and the objective is to minimize the makespan of all jobs, i.e. finish all jobs as soon as possible.

4.1.1 Implementation Components

MDP Formulation. As shown in Fig. 2(a), *state* is the current DAG $\mathcal{G}^k$, *action* is defined as adding an edge to $\mathcal{G}^k$, resulting in a new DAG $\mathcal{G}^{k+1}$. The added edge enforces additional constraints such that the decision space is cut down to elevate the heuristic's performance. Here $\mathbf{x}$ represents the scheduled execution order, and the Critical Path heuristic is adopted on $\mathcal{G}^{k+1}$ to compute $\mathbf{x}^{k+1}$. The *reward* is computed by subtracting the previous makespan by the new makespan: $f(\mathbf{x}^k|\mathcal{G}) - f(\mathbf{x}^{k+1}|\mathcal{G})$.

State Encoder. We adopt GCN [30] to encode the state represented by DAG. Considering the structure of DAG, we design two GCNs: the first GCN processes the original DAG, and the second GCN processes the DAG with all edges reversed. The node embeddings from two GCN modules are concatenated, and an attention pooling layer is adopted to extract a graph-level embedding.

$$\mathbf{n} = \left[\text{GCN}_1(\mathcal{G}^k) \,||\, \text{GCN}_2(\text{reverse}(\mathcal{G}^k))\right],\ \mathbf{g} = \text{Att}(\mathbf{n}) \tag{3}$$

where $\mathbf{n}$ (# of nodes $\times$ feature dim) is the node feature, $[\cdot \,||\, \cdot]$ means concatenating two features.

Actor Net. To reduce action space, the edge selection is decomposed into two steps: first selecting the starting node and then the ending node. The action probabilities of selecting the starting and ending nodes are predicted by two independent 3-layer ResNets blocks [20], respectively, and the input of the second ResNet contains additionally the feature vector of the selected starting node.

$$P(\mathbf{a}_1) = \text{softmax}(\text{ResNet}_1([\mathbf{n} \,||\, \mathbf{g}])),\ P(\mathbf{a}_2|\mathbf{a}_1) = \text{softmax}(\text{ResNet}_2([\mathbf{n} \,||\, \mathbf{n}[\mathbf{a}_1] \,||\, \mathbf{g}])) \tag{4}$$

Table 1: Experimental results on DAG scheduling problems from TPC-H dataset with the average number of nodes reported in brackets. The dataset name TPC-H-X means X jobs are jointly scheduled. The upper half is learning-free heuristics, and the lower half is RL-based methods where PPO-Single can be viewed as our implementation of the peer single-level RL methods [36, 60]. Note "objective" means the objective score i.e. the total makespan time (in seconds, the smaller the better) to finish all jobs, and "relative" is computed by the solved makespan time w.r.t. Critical Path heuristic.

dataset / method	TPC-H-50 (#nodes=467.2)		TPC-H-100 (#nodes=929.8)		TPC-H-150 (#nodes=1384.5)	
	objective ↓	relative ↓	objective ↓	relative ↓	objective ↓	relative ↓
Shortest Job First	12818	30.5%	19503	15.3%	27409	12.2%
Tetris [18]	12113	23.3%	18291	8.1%	25325	3.7%
Critical Path	9821	0.0%	16914	0.0%	24429	0.0%
PPO-Single [28, 36]	10578	7.7%	17282	2.2%	24822	1.6%
PPO-BiHyb (ours)	**8906**	**-9.3%**	**15193**	**-10.2%**	**22371**	**-8.4%**

the subscript of $\mathbf{n}[\mathbf{a}_1]$ denotes the embedding for a node $\mathbf{a}_1$. For training, we sample $\mathbf{a}_1, \mathbf{a}_2$ according to their probabilities $P(\mathbf{a}_1), P(\mathbf{a}_2|\mathbf{a}_1)$ respectively. For testing, beam search is performed with a width of 3: actions with top-3 highest probabilities are searched, and all searched actions are evaluated by the improvement of makespan, and only the top-3 actions are maintained for the next search step.

Critic Net. It is built by max pooling over all node features, and the pooled feature is concatenated with the graph feature from state encoder, and finally processed by another ResNet for value prediction.

$$\widetilde{V}(\mathcal{G}^k) = \text{ResNet}_3\left([\text{maxpool}(\mathbf{n}) \,||\, \mathbf{g}]\right) \tag{5}$$

Heuristic Methods. Solving DAG scheduling with hundreds of nodes is nearly infeasible for existing commercial solvers, and real-world scheduling problems are usually tackled by fast heuristics e.g. *Shortest Job First* which schedules the shortest job greedily and *Critical Path* algorithm which prioritizes jobs on the critical path. We also consider the novel *Tetris* scheduling [18] where jobs are arranged as a Tetris game on the two-dimension space of makespan and resource. Since *Critical Path* is empirically effective, we use it as the lower level optimization algorithm in our PPO-BiHyb learning method.

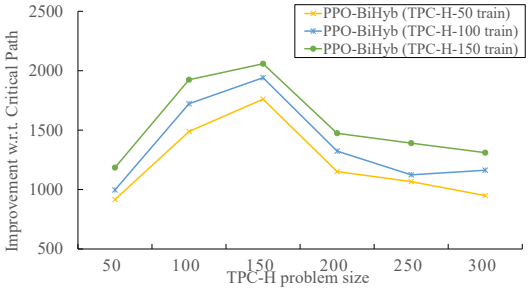


Figure 3: Generalization result on TPC-H dataset.

Learning Methods. There are some efforts to train a scheduler in data centers with RL [36] and learning a job shop scheduler with RL [60]. They can be viewed as embodiments with tailored techniques for specific problems based on the end-to-end single-level pipeline [28]. Since the problem settings in [36, 60] are different with ours, we re-implement the single-level RL baseline PPO-Single following [28], where the model details are in line with our PPO-BiHyb.

4.1.2 Experimental Results

Results are reported for scheduling jobs from TPC-H dataset[1], which is composed of business-oriented queries and concurrent data modifications, represented by Directed Acyclic Graphs (DAGs). Each DAG represents a unique computation job, and each node in DAG has two properties: execution time and resource requirement. We jointly schedule multiple DAGs and assume the heaviest job node consumes up to 10% of total resources, to reflect real-world business scenarios. We build 50 training and 10 testing samples for all experiments, where each sample is composed of a number of DAGs (e.g. 50, 100, 150) which are randomly sampled from the TPC-H dataset with fixed random seed.

Table 1 reports the result on scheduling jobs from TPC-H dataset, where our PPO-BiHyb outperforms learning-free heuristics and the single level learning method. We control the beam search width of PPO-Single as 10 allowing its inference time to be $10\times$ longer than ours, but its performance is still inferior to the heuristic Critical Path method. We also study the generalization of our learning approach, which is critically important for real-world applications. As shown in Fig. 3, our models are learned with TPC-H-50/100/150 datasets, and we report their test results on the unseen datasets

[1] `http://tpc.org/tpch/default5.asp`

ranging from TPC-H-50 to 300 at 50 interval. The plotted results are the improvement of makespan w.r.t. Critical Path. All our learning models generalize well with unseen data, even on larger problems with up to 300 DAGs ($\sim$2800 nodes). The generalization results of PPO-Single are omitted because they are consistently inferior to the Critical Path heuristic. The reason that our method generalizes soundly, in our analysis, is because learning the graph-optimization strategy is easier compared to directly learning the solution, and the strength of heuristics is exploited with our approach.

4.2 Case 2: Graph Edit Distance

The graph edit distance (GED) problem is NP-complete [1] and can be readily used in modeling a family of pattern recognition tasks requiring to measure the similarity between graphs, with applications to drug discovery [40], malware detection [32] and scene graph edition [7]. Given two graphs, the objective is finding the cheapest edit path from one graph to the other, and the costs of edit operations are defined by problem-specific distance metrics. For example, for molecule graphs, we may define an edit cost between different atom types (i.e. different node types), and also an edit cost for creating/removing a chemical bound (i.e. adding/removing an edge).

4.2.1 Implementation Components

MDP Formulation. As shown in Fig. 2(b), since there are two graphs, we discriminate the graphs by subscripts $\mathcal{G}_1, \mathcal{G}_2$, and the MDP formulation is a generalized version of Sec. 3.2.1. More specifically, the RL agent is restricted to modifying $\mathcal{G}_1$ and keeping $\mathcal{G}_2$ fixed. The *state* is defined as the current version of first graph $\mathcal{G}_1^k$, and *action* is defined as adding or removing an edge from $\mathcal{G}_1^k$. The new graph $\mathcal{G}_1^{k+1}$ is designed to better align with $\mathcal{G}_2$, where the node-to-node alignment is encoded by the decision variable $\mathbf{x}^k$ which is obtained by solving $f(\mathbf{x}^k|\mathcal{G}_1^k, \mathcal{G}_2)$ with IPFP heuristic. The upper-level objective $f(\mathbf{x}^k|\mathcal{G}_1, \mathcal{G}_2)$ is computed as the edit distance between the original graphs $\mathcal{G}_1, \mathcal{G}_2$ using $\mathbf{x}^k$.

State Encoder. The state encoder is built with GCN [30] to aggregate graph features and Sinkhorn-Knopp (SK) network [46] for propagation across two graphs. Following [55], the SK module accepts a similarity matrix (by inner-product of $\mathbf{n}_1$ and $\mathbf{n}_2$), and outputs a doubly-stochastic matrix which can be utilized as the cross-graph propagation weights. As the action is taken on the first graph, we compute the difference of the node features of graph 1 and the propagated features from graph 2 through SK net. Similar to the DAG model, graph-level features are obtained via attention pooling.

$$\mathbf{n}_1 = \text{GCN}(\mathcal{G}_1^k),\ \mathbf{n}_2 = \text{GCN}(\mathcal{G}_2),\ \mathbf{n} = \mathbf{n}_1 - \text{SK}(\mathbf{n}_1\mathbf{n}_2^\top) \cdot \mathbf{n}_2;\ \mathbf{g}_1 = \text{Att}(\mathbf{n}_1),\ \mathbf{g}_2 = \text{Att}(\mathbf{n}_2) \quad (6)$$

Actor Net. The action of selecting an edge to add or delete is also decomposed by two node-selection steps. The starting node selection is predicted by 3-layer ResNet module, and the ending node selection is predicted by an attention query. The edge is deleted if it already exists, or added otherwise. The beam width is set as 3 during evaluation.

$$P(\mathbf{a}_1) = \text{softmax}\left(\text{ResNet}(\mathbf{n})\right),\ P(\mathbf{a}_2|\mathbf{a}_1) = \text{softmax}(\mathbf{n} \cdot \tanh(\text{Linear}\left(\mathbf{n}[\mathbf{a}_1]\right))^\top) \quad (7)$$

Critic Net. We follow the graph-wise similarity learning method [3] to implement the critic net, where the graph-level features are processed by a neural tensor network (NTN) [47] followed by 2 fully-connected regression layers whose output is one-dimensional:

$$\widetilde{V}(\mathcal{G}_1^k, \mathcal{G}_2) = \text{fc}\left(\text{NTN}\left(\mathbf{g}_1, \mathbf{g}_2\right)\right) \quad (8)$$

Heuristic Methods. Based on the comprehensive evaluation on different GED heuristics by [5], we select 4 best-performing heuristics: *Hungarian* [41] which simplifies the original problem to a bipartite matching problem, *RRWM* [9] which solves GED via relaxed quadratic programming, *Hungarian-Search* [42] which is a search algorithm guided by Hungarian heuristic, and *IPFP* [6] which combines searching and quadratic programming. We empirically find *IPFP* best performs among all heuristics, therefore we base our PPO-BiHyb method on it.

Learning Methods. There are efforts to learn graph-wise similarity via deep learning [3, 32], which are regression models and ignores the combinatorial nature of graph similarity problems. [57] proposes neural-guided A* search, however, the learning is supervised. In this paper, we compare with PPO-Single by reimplementing [33] for GED. The model details and RL algorithm are kept in line with our PPO-BiHyb for fair comparison. The beam search width of PPO-Single is set to 20 so that the inference time of PPO-Single is comparable with our hybrid method.

Table 2: Results on graph edit distance (GED) problems from AIDS dataset with the average number of nodes reported in brackets. The naming convention here AIDS-X/Y means the number of nodes are within the range of X and Y. PPO-Single can be viewed as our implementation of the peer RL method [33]. Here "objective" means the objective score i.e. the solved edit distance between two graphs, and "relative" is computed by the solved edit distance w.r.t. the best heuristic IPFP.

method \ dataset	AIDS-20/30 (#nodes=22.6)		AIDS-30/50 (#nodes=37.9)		AIDS-50+ (#nodes=59.6)	
	objective ↓	relative ↓	objective ↓	relative ↓	objective ↓	relative ↓
Hungarian [41]	72.9	94.9%	153.4	117.9%	225.6	121.4%
RRWM [9]	72.1	92.8%	139.8	98.6%	214.6	110.6%
Hungarian-Search [42]	44.6	19.3%	103.9	47.6%	143.8	41.1%
IPFP [6]	37.4	0.0%	70.4	0.0%	101.9	0.0%
PPO-Single [33]	56.5	51.1%	110.0	56.3%	183.9	80.5%
PPO-BiHyb (ours)	**29.1**	**-22.2%**	**61.1**	**-13.2%**	**77.0**	**-24.4%**

4.2.2 Experimental Results

Results on GED are reported on AIDS dataset[2] containing chemical compounds for anti-HIV research [40]. The atoms are treated as nodes, and the atom types are encoded by one-hot features and we define an edit cost of 1 for different node types. Besides, we also define the cost for node/edge addition/deletion as 1. The AIDS dataset is split into three subsets w.r.t. the size of graphs, namely AIDS-20/30, AIDS-30/50, and AIDS-50+, and we exclude graphs smaller than 20 nodes because they are less challenging and can be solved exactly within several hours. We randomly build 50 training and 10 testing samples for all tests with fixed random seed.

The evaluation for GED on AIDS is presented in Tab. 2, where our method surpasses all learning-free heuristics and also performs better than single-level RL baseline PPO-Single. We also conduct a generalization study among different problem sizes, and Fig. 4 shows the objective scores from the sensitivity test for training/testing on GED problems with different sizes. The color map represents the percentage of improvement w.r.t. IPFP, where the darker color means more improvement. Our model can generalize to problem sizes unseen during training, but the generalized performance is usually inferior compared to training and testing with the same problem size.

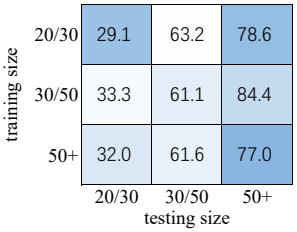


Figure 4: Sensitivity by training and testing sizes on AIDS.

4.3 Case 3: Hamiltonian Cycle Problem

The Hamiltonian cycle problem (HCP) arises from the notable problem of seven bridges of Königsberg proposed by Leonhard Eule [12]. Given a graph, the HCP arises as a decision problem on whether there exists a Hamiltonian cycle, which is known to be NP-complete [15]. We handle HCP by transforming it into the more general traveling salesman problem (TSP) in a fully-connected graph, where the existing edges in HCP are defined with length 0, and non-existing edges are with length 1. If a tour is found whose length is 0, then it is a Hamiltonian cycle. It is worth noting that HCP has not been discussed by most ML-TSP works [14, 26, 31, 54], which focus on the special case of Euclidean TSP with 2D coordinates.

4.3.1 Implementation Components

MDP Formulation. The HCP instances are converted to TSP as discussed above, to leverage existing powerful TSP heuristics. As shown in Fig. 2(c), *state* is defined as the current graph $\mathcal{G}^k$ and *action* is defined as increasing an edge length in $\mathcal{G}^k$, resulting in a new graph $\mathcal{G}^{k+1}$. The increase of edge length softly prohibits the heuristic from traveling this edge and the LKH method [22] is adopted on graph $\mathcal{G}^{k+1}$ to obtain the new tour $\mathbf{x}^{k+1}$. The *reward* is the decrease of the current tour length w.r.t. the previous tour length: $f(\mathbf{x}^k|\mathcal{G}) - f(\mathbf{x}^{k+1}|\mathcal{G})$.

State Encoder. We encode HCP graph with GCN [30]. The node embeddings from GCN modules are then processed by an attention pooling layer to extract a graph-level embedding.

$$\mathbf{n} = \text{GCN}(\mathcal{G}^k),\ \mathbf{g} = \text{Att}(\mathbf{n}) \tag{9}$$

[2] https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data

Actor Net. After obtaining the tour solved by the heuristic algorithm, we increase the length of any selected edge by 10 from the tour (10 is randomly set, and our approach seems not sensitive to this number), which empirically makes the edge harder to be selected in a tour later. The first action probabilities of selecting the starting node are predicted by a 3-layer ResNet block [20]. The second action is selecting the ending node, which is adjacent to the starting node on the tour and is predicted by an attention query.

$$P(\mathbf{a}_1) = \text{softmax}(\text{ResNet}_1([\mathbf{n} \,||\, \mathbf{g}])),\ P(\mathbf{a}_2|\mathbf{a}_1) = \text{softmax}\left(\mathbf{n} \cdot \tanh(\text{Linear}(\mathbf{n}[\mathbf{a}_1]))^\top\right) \tag{10}$$

For training, we sample $\mathbf{a}_1, \mathbf{a}_2$ according to $P(\mathbf{a}_1), P(\mathbf{a}_2|\mathbf{a}_1)$. For evaluation, we perform a beam search with a width of 12 and maintain the top-12 actions by the improvement of the LKH algorithm.

Critic Net. It is built by max-pooling from all node features, and the pooled feature is concatenated with the graph-level feature from the state encoder, which are finally processed by a 3-layer ResNet.

$$\widetilde{V}(\mathcal{G}^k) = \text{ResNet}_2\left([\text{maxpool}(\mathbf{n}) \,||\, \mathbf{g}]\right) \tag{11}$$

Heuristic Methods. The performance of heuristics varies with the sizes and characteristics of instances. We choose three algorithms, *Nearest Neighbour* [39] who greedily travels to the next nearest node, *Farthest Insertion* [43] who repeatedly insert the non-traveled node with the farthest distance to the existing tour, and the third version of *Lin-Kernighan Heuristic (LKH3)* [22] which succeeds in discovering the best-known solutions for many TSP instances. For LKH3, we can set the number of random restarts (5 by default) to trade-off time for accuracy. We name the default LKH config as *LKH3-fast* which is also adopted for lower-level optimization in our PPO-BiHyb for its time-efficiency, and we also compare with *LKH3-accu* with 100 random restarts.

Table 3: Tests on FHCP with mean number of nodes in brackets. FHCP-`X`/`Y` means the number of nodes is in the range of `X` and `Y`.

dataset / method	FHCP-500/600 (#nodes=535.1)	
	TSP objective ↓	found cycles ↑
Nearest Neighbor [39]	79.6	0%
Farthest Insertion [43]	133.0	0%
LKH3-fast [22]	13.8	0%
LKH3-accu [22]	**6.3**	20%
PPO-Single [28]	465.65	0%
PPO-BiHyb (ours)	8.5	**25%**

Learning Methods. Most deep learning TSP models are designated to handle fully connected 2D Euclidean TSP [14, 26, 31, 54], which is a different setting compared to our HCP testbed. Therefore, we implement PPO-Single following the most general framework [28], and the model details are kept in line with our PPO-BiHyb.

4.3.2 Experimental Results

We use the FHCP benchmark [19] composed of 1001 hard HCP instances. All instances are known to have valid Hamiltonian cycles, however, finding them is non-trivial for standard HCP/TSP heuristics. Table 3 shows the result from a subset of FHCP benchmark. As mentioned in Sec. 4.3.1, we convert HCP instances to binary TSP instances, whose tour lengths are denoted as "TSP objective". When the TSP objective is 0, it means that a Hamiltonian cycle is found. We use 50 training instances and 20 testing instances, and the model is trained on graphs of sizes from 250 to 500. Our method is comparative with the novel heuristic *LKH3-accu* and can even surpass in terms of found Hamiltonian cycles, which is the objective of HCP. We set the beamwidth of PPO-Single as 12 allowing its inference time to be $10\times$ longer than ours, but its performance is still inferior to most heuristics.

5 Discussions

Limitations. Our proposed method may not fit problems if narrowing down the feasible space cannot be achieved via graph modification, e.g. the expression simplification problem. Besides, the experimented datasets may not fully reflect the challenges in real-world scenarios.

Potential negative impacts. Our approach may potentially decrease the job opportunities and burden the workload of employees in companies, and it calls for the companies and social groups to take more responsible roles when facing the negative effects that come along with optimization tools.

Conclusion. We present a bi-level optimization framework based on hybrid machine learning and traditional heuristics, where the graph structure is optimized by RL to narrow down the feasible space of combinatorial problems. The new optimization problems are solved by fast heuristics. Experiments on large real-world combinatorial problems show the effectiveness of our approach. Our method also shows good generalization ability from small training instances to larger testing instances.

References

[1] Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, and P. Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In *Int. Conf. Pattern Recog.*, 2015.

[2] S. A. Bagloee, M. Asadi, M. Sarvi, and M. Patriksson. A hybrid machine-learning and optimization method to solve bi-level problems. *Expert Systems with Applications*, 95:142–152, 2018.

[3] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 384–392, 2019.

[4] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

[5] D. B. Blumenthal, N. Boria, J. Gamper, S. Bougleux, and L. Brun. Comparing heuristics for graph edit distance computation. *The VLDB journal*, 29(1):419–458, 2020.

[6] S. Bougleux, B. Gaüzere, and L. Brun. Graph edit distance as a quadratic program. In *Int. Conf. Pattern Recog.*, pages 1701–1706. IEEE, 2016.

[7] L. Chen, G. Lin, S. Wang, and Q. Wu. Graph edit distance reward: Learning to edit scene graph. *Eur. Conf. Comput. Vis.*, 2020.

[8] X. Chen and Y. Tian. Learning to perform local rewriting for combinatorial optimization. *Neural Info. Process. Systems*, 32:6281–6292, 2019.

[9] M. Cho, J. Lee, and K. M. Lee. Reweighted random walks for graph matching. In *Eur. Conf. Comput. Vis.*, pages 492–505. Springer, 2010.

[10] H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. In *Int. Conf. Mach. Learn.*, pages 2702–2711. PMLR, 2016.

[11] L. Duan, H. Hu, Y. Qian, Y. Gong, X. Zhang, J. Wei, and Y. Xu. A multi-task selected learning approach for solving 3d flexible bin packing problem. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1386–1394, 2019.

[12] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

[13] A. Forti. *DAG Scheduling for Grid Computing Systems*. PhD thesis, University of Udine, Italy, 2006.

[14] Z.-H. Fu, K.-B. Qiu, and H. Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. *AAAI Conf. Artificial Intell.*, 2021.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. 1990.

[16] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Neural Info. Process. Systems*, 2019.

[17] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem. In *50 Years of Integer Programming 1958-2008*, pages 77–103. Springer, 2010.

[18] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. *SIGCOMM Comput. Commun. Rev.*, 44(4):455–466, Aug. 2014.

[19] M. Haythorpe. Fhcp challenge set: The first set of structurally difficult instances of the hamiltonian cycle problem. *arXiv preprint arXiv:1902.10352*, 2019.

[20] K. He, G. Gkioxari, P. Dollár, and B. R. Girshick. Mask r-cnn. *Int. Conf. Comput. Vis.*, pages 386–397, 2017.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Comput. Vis. Pattern Recog.*, pages 770–778, 2016.

[22] K. Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[23] K. Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 2017.

[24] R. Hu, J. Xu, B. Chen, M. Gong, H. Zhang, and H. Huang. Tap-net: transport-and-pack using reinforcement learning. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[25] R. G. Jeroslow. The polynomial hierarchy and a simple model for competitive analysis. *Mathematical programming*, 32(2):146–164, 1985.

[26] C. K. Joshi, T. Laurent, and X. Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

[27] N. Karalias and A. Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Neural Info. Process. Systems*, 2020.

[28] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In *Neural Info. Process. Systems*, pages 6351–6361, 2017.

[29] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina. Learning to branch in mixed integer programming. In *AAAI Conf. Artificial Intell.*, volume 30, 2016.

[30] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *Int. Conf. Learn. Rep.*, 2017.

[31] W. Kool, H. van Hoof, and M. Welling. Attention, learn to solve routing problems! In *Int. Conf. Learn. Rep.*, pages 1–25, 2019.

[32] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International Conference on Machine Learning*, pages 3835–3845, 2019.

[33] C. Liu, R. Wang, Z. Jiang, and J. Yan. Deep reinforcement learning of graph matching. *arXiv preprint arXiv:2012.08950*, 2020.

[34] R. Liu, J. Gao, J. Zhang, D. Meng, and Z. Lin. Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond. *arXiv preprint arXiv:2101.11517*, 2021.

[35] H. Lu, X. Zhang, and S. Yang. A learning-based iterative method for solving vehicle routing problems. In *Int. Conf. Learn. Rep.*, 2019.

[36] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 270–288, 2019.

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[38] A. Nowak, S. Villar, A. Bandeira, and J. Bruna. Revised note on learning quadratic assignment with graph neural networks. In *Data Science Workshop*, 2018.

[39] G. Reinelt. *The traveling salesman: computational solutions for TSP applications*, volume 840. Springer, 2003.

[40] K. Riesen and H. Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 287–297. Springer, 2008.

[41] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959, 2009.

[42] K. Riesen, S. Fankhauser, and H. Bunke. Speeding up graph edit distance computation with a bipartite heuristic. In *Mining and Learning with Graphs*, pages 21–24, 2007.

[43] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, 6(3):563–581, 1977.

[44] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Int. Conf. Mach. Learn.*, pages 1889–1897. PMLR, 2015.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[46] R. Sinkhorn and A. Rangarajan. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Ann. Math. Statistics*, 1964.

[47] R. Socher, D. Chen, C. D. Manning, and A. Y. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Neural Info. Process. Systems*, page 926–934, Red Hook, NY, USA, 2013. Curran Associates Inc.

[48] H. Sun, W. Chen, H. Li, and L. Song. Improving learning to branch via reinforcement learning. *NeurIPS Workshop*, 2020.

[49] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[50] E.-G. Talbi. *A Taxonomy of Metaheuristics for Bi-level Optimization*, pages 1–39. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[51] Y. Tang, S. Agrawal, and Y. Faenza. Reinforcement learning for integer programming: Learning to cut. In *Int. Conf. Mach. Learn.*, pages 9367–9376. PMLR, 2020.

[52] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[53] L. Vicente, G. Savard, and J. Júdice. Descent approaches for quadratic bilevel programming. *Journal of Optimization Theory and Applications*, 81(2):379–399, 1994.

[54] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Neural Info. Process. Systems*, pages 2692–2700, 2015.

[55] R. Wang, J. Yan, and X. Yang. Combinatorial learning of robust deep graph matching: an embedding based approach. *IEEE TPAMI*, 2020.

[56] R. Wang, J. Yan, and X. Yang. Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *Trans. Pattern Anal. Mach. Intell.*, 2021.

[57] R. Wang, T. Zhang, T. Yu, J. Yan, and X. Yang. Combinatorial learning of graph edit distance via dynamic embedding. *Comput. Vis. Pattern Recog.*, 2021.

[58] J. Yan, S. Yang, and E. R. Hancock. Learning for graph matching and related combinatorial optimization problems. In *Int. Joint Conf. Artificial Intell.*, 2020.

[59] Y. Yang and A. B. Whinston. A survey on reinforcement learning for combinatorial optimization. *CoRR*, abs/2008.12248, 2020.

[60] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Neural Info. Process. Systems*, 33, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See discussions in Section 5 and Appendix A.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See discussions in Section 5 and Appendix F.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Sec. 3.1.
   (b) Did you include complete proofs of all theoretical results? [Yes] See the proposition and proof in Sec. 3.1.
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code is proprietary and requires to undergo some internal processes before being made public. We will release the code if this paper is accepted. The dataset links and citations are provided.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See experiment details in Sec. 4 and Appendix C.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] As discussed in Sec. 4.1.2 and 4.2.2, random seeds are fixed to ensure that all experiments are run with the same training/testing instances, else the reported objective scores will be different simply due to different training/testing instances in different runs.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix E.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes] See Appendix G.

(c) Did you include any new assets either in the supplemental material or as a URL? [No] We do not include any new assets.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] See Appendix G and all assets are publicly available.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] See Appendix G and all assets do not contain human information or offensive content.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We do not use crowdsourcing or conduct research with human subjects.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]