DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recently, deep reinforcement learning (DRL) models have shown promising results in solving NP-hard Combinatorial Optimization (CO) problems. However, most DRL solvers can only scale to a few hundreds of nodes for combinatorial optimization problems on graphs, such as Traveling Salesman Problem (TSP). This paper addresses the scalability challenge in large-scale combinatorial optimization by proposing a novel approach, namely, DIMES. Unlike previous DRL methods which suffer from costly autoregressive decoding or iterative refinements of discrete solutions, DIMES introduces a compact continuous space for parameterizing the underlying distribution of candidate solutions. Such a continuous space allows stable REINFORCE-based training and fine-tuning via massively parallel sampling. We further propose a meta-learning framework to enable effective initialization of model parameters in the fine-tuning stage. Extensive experiments show that DIMES outperforms recent DRL-based methods on large benchmark datasets for Traveling Salesman Problems and Maximal Independent Set problems.

1 Introduction

Combinatorial Optimization (CO) is a fundamental problem in computer science. It has important real-world applications such as shipment planning, transportation, robots routing, biology, circuit design, and more [54]. However, due to the NP-hard complexity, a significant portion of the CO problems suffer from an exponential computational cost when using traditional algorithms. As a well-known example, the Travelling Salesman Problem (TSP) has been intensively studied [28, 48] for finding the most cost-effective tour over an input graph where each node is visited once and only once before finally returning to the starting node. Over the past decades, significant efforts has been made for designing more efficient heuristic solvers [5, 17] for approximating the solution in a reduced search space.

Recent development in deep reinforcement learning (DRL) have shown promises in solving CO problems without manual injection of domain-specific expert knowledge [7, 35, 38]. The appeal of the neural methods is that they can learn useful patterns (such as graph motifs) from data that might be difficult to discover by hand. A typical category of DRL solvers, namely the *construction heuristics learners*, [7, 35] use a Markov decision process (MDP) to grow partial solutions by adding one new node per step, with a trained strategy which gives good solutions higher probabilities. Another category of DRL-based solvers, namely the *improvement heuristics learners* [10, 59], iteratively refine a feasible solution with a neural network-guided local Operations Research (OR) operations [51]. A major limitation of those DRL solvers is in their scalability on large CO problems. For example, current DRL solvers for TSP can only scale to graphs with up to hundreds of nodes.

The bad scalability of those DRL methods lies on the fact that they suffer from the costly decoding of CO solutions, which is typically linear in the number of nodes in the input graph. Since the reward

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

of reinforcement learning is determined after decoding a complete solution (with either a chain rule factorization or iterative refinements), either construction or improvement heuristic learners would encounter the sparse reward problem when dealing with large graphs [35, 26, 31]. While such an overhead can be partially alleviated by constructing several parts of the solution in parallel [1] for locally decomposable CO problems[1], such as for maximum independent set (MIS) problems [45], how to scale up neural solvers for CO problems in general, including the locally non-decomposable ones (such as TSP) is still an open challenge.

In this paper, we address this challenge by proposing a novel framework, namely DIMES (DIfferentiable MEta Solver) for solving combinatorial optimization problems. Instead of previous DIR-based CO sovles which rely on construction or improvement heuristics, we introduce a compact continuous space for parameterizing the underlying distribution of candidate solutions which allows massively parallel on-policy sampling without the costly decoding process, and effectively reduces the variance of the gradients by the REINFORCE algorithm [58] during both training and fine-tuning phases. We further propose a meta-learning framework to enable effective initialization of model parameters in the fine-tuning stage. To our knowledge, we are the first to apply meta-learning over a collection of CO problem instances, where each instance graph is treated as one of a collection tasks in a unified framework.

We need to point out that the idea of designing a continuous space for combinatorial optimization problems has been tried by the heatmaps approaches in the literature [41, 24, 16, 37]. But there are several major distinctions between the existing methods and our proposed one. The prior works generate heatmaps based on supervised signals (each training graph is paired with its best solution) [4, 18], which are costly to obtain. DIMES is directly optimized with gradients estimated by the REINFORCE algorithm, which do not require supervised signals. As a result, DIMES can scale to large graphs with up to tens of thousands of nodes, and predict (nearly) optimal solutions without the needs for costly generation of supervised training data or human specification of problem-specific heuristics.

In our experiments, we show that DIMES outperforms strong baselines among DRL-based solvers on TSP benchmark datasets, and can be successfully scaled up to graphs with tens of thousands of nodes. As a sanity check, we also evaluate our framework on locally decomposable combinatorial optimization problems, including Maximal Independent Set (MIS) problem for synthetic graphs and graphs reduced from satisfiability (SAT) problems. Our experimental results show that DIMES achieve competitive performance compared with neural solvers that are specially designed for locally decomposable CO problems.

2 Related Work

2.1 DRL-based Construction Heuristics Learners

Construction heuristics methods create a solution of CO problem instance in one shot without further modifications. Bello et al. [7] are the first to tackle combinatorial optimization problems using neural networks and reinforcement learning. They used a Pointer Network (PtrNet) [55] as the policy network and employed the actor-critic algorithm [33] for training on TSP and KnapSack instances. Further improved models have been developed afterwards [13, 35, 50, 39], such as attention models [53], better DRL algorithms [30, 43, 36, 38, 47, 60, 56], for an extended scope of CO problems such as Capacitated Vehicle Routing Problem (CVRP) [46], Job Shop Scheduling Problem (JSSP) [63], Maximal Independent Set (MIS) problem [29, 1], and boolean satisfiability problem (SAT) [62].

Our proposed method in this paper belongs to the category of *construction heuristics leaners* in the sense of producing a one-shot solution per problem instance. But unlike previous methods which generate the solutions via a constructive Markov decision process (MDP) with rather costly decoding steps (adding one un-visited node per step to a partial solution), we introduce a compact continues space to parameterize the underlying distribution of discrete candidate solutions, and to allow efficient sampling from such distribution without costly neural network-involved decoding.

[1] Locally decomposable problem refers to the problem where the feasibility constraint and the objective can be decomposed by locally connected variables (in a graph) [1].

2.2 DRL-based Improvement Heuristics Learners

Contrast to construction heuristics, DRL-based improvement heuristics methods train a neural network to iteratively correct the mistakes locally for improving the quality of the current solution until the time budget runs out. Such DRL-based improvement heuristics methods are usually inspired by classical local search algorithms such as 2-opt [11] and the large neighborhood search (LNS) [52], and have been demonstrated with outstanding results by many [59, 42, 57, 12, 10, 21, 61, 44, 23, 31]. Improvement heuristics methods generally show better performance than construction heuristics but are slower in computation in return.

2.3 Supervised Learners for CO Problems

Vinyals et al. [55] trained a Pointer Network to predict a TSP solution based on supervised signals from the Held-Karp algorithm [6] or approximate algorithms. Li et al. [41] and Joshi et al. [24] trained a graph convolutional network to predict the possibility of each node or edge to be included in the optimal solutions of MIS and TSP problems, respectively. Recently, Joshi et al. [25] showed that unsupervised reinforcement learning leads to better emergent generalization over various sized graphs than supervised learning. Our work in this paper provides further evidence for the benefits of the unsupervised training, or more specifically, unsupervised generation of heatmaps [41, 24, 16, 37], for combinatorial optimization problems.

3 Proposed Method

3.1 Formal Definitions

Following a conventional notation [49] we define $\mathcal{F}_s$ as the set of discrete feasible solutions for a CO problem instance s, and $c_s : \mathcal{F}_s \to \mathbb{R}$ as the cost function over any element $f \in \mathcal{F}_s$. The objective is to find the optimal solution as:

$$f_s^* = \operatorname*{argmin}_{f \in \mathcal{F}_s} c_s(f), \tag{1}$$

For the Traveling Salesman Problem (TSP), $\mathcal{F}_s$ is the set of all the routes that visits each node exactly once and returns to the starting node by the end, and c_s calculates the cost for each route $f \in \mathcal{F}_s$ by summing up the edge weights in the route. The size of $\mathcal{F}_s$ for TSP is $n!$ for a graph with n nodes. For the Maximal Independent Set (MIS) problem, $\mathcal{F}_s$ is a subset of the power set $\mathcal{S}_s = \{0, 1\}^n$ and consists of all the independent subsets where each node a subset has no connection to any other node in the same subset, and c_s calculates the negation of the size of each independent subset.

We parameterize the solution space with a continuous and differentiable vector $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{V}_s|}$, where $\mathcal{V}_s$ denotes the variables in the problem instance s (e.g., edges in TSP and nodes in MIS), and estimates the probability of each feasible solution f as:

$$p_{\boldsymbol{\theta}}(f \mid s) \propto \exp(\sum_{i=1}^{|\mathcal{V}_s|} f_i \cdot \theta_i) \quad \text{subject to} \quad f \in \mathcal{F}_s. \tag{2}$$

where $p_{\boldsymbol{\theta}}$ is an energy function over the discrete feasible solution space, f is a $|\mathcal{V}_s|$-dimensional vector with element $f_i \in \{0, 1\}$ indicating whether the ith variable is included in feasible solution f, and the higher value of θ_i means a higher probability for the ith variable produced by $p_{\boldsymbol{\theta}}(f \mid s)$.

3.2 Gradient-based Optimization

When the combinatorial problem is locally decomposable, such a MIS, a penalty loss [27, 2] can be added to suppress the unfeasible solutions as:

$$\ell_{\text{Erdos}}(\boldsymbol{\theta} \mid s) = \sum_{f \in \mathcal{S}_s} \left[p_{\boldsymbol{\theta}}(f \mid s) \cdot (c_s(f) + \beta \cdot \mathbb{1}(f \notin \mathcal{F}_s)) \right]. \tag{3}$$

where $\beta > \max_{f \in \mathcal{F}_s} c_s(f)$. The objective function ℓ_{Erdos} can thus be calculated analytically and enable end-to-end training. However, this is not possible for more structured combinatorial problems

such as TSP[2]. Therefore, we propose to directly optimize the expected cost over the underlying population of feasible solutions, which is defined as:

$$\ell_p(\boldsymbol{\theta} \mid s) = \mathbb{E}_{f \sim p_{\boldsymbol{\theta}}} \left[c_s(f) \right]. \tag{4}$$

Optimizing this objective requires efficient sampling, with which REINFORCE-based [58] gradient estimation can be calculate. Nevertheless, a common practice to sample from the energy $p_{\boldsymbol{\theta}}$ functions requires MCMC [40], which is not efficient enough. So we propose to design an auxiliary distribution $q_{\boldsymbol{\theta}}$ over the feasible solutions $\mathcal{F}_s$, such that the following conditions hold: 1) sampling from $q_{\boldsymbol{\theta}}$ is efficient, and 2) $q_{\boldsymbol{\theta}}$ and $p_{\boldsymbol{\theta}}$ should convergence to the same optimal $\boldsymbol{\theta}^*$. Then, we can replace $p_{\boldsymbol{\theta}}$ by $q_{\boldsymbol{\theta}}$ in our objective function as:

$$\ell_q(\boldsymbol{\theta} \mid s) = \mathbb{E}_{f \sim q_{\boldsymbol{\theta}}} \left[c_s(f) \right], \tag{5}$$

and get the REINFORCE-based update rule as:

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{f \sim q_{\boldsymbol{\theta}}} \left[c_s(f) \right] = \mathbb{E}_{f \sim q_{\boldsymbol{\theta}}} (c_s(f) - b(s)) \nabla_{\boldsymbol{\theta}} \log q_{\boldsymbol{\theta}}(f), \tag{6}$$

where $b(s)$ denotes a baseline function that does not depend on f and estimates the expected cost to reduce the variance of the gradients. In this paper, we use a sampling-based baseline function proposed by Kool et al. [36].

Next, we specify the auxiliary distributions for TSP and MIS, respectively. For brevity, we omit the conditional notations of s for all probability formulas in the rest of the paper.

3.2.1 Auxiliary Distribution for TSP

For the TSP problem for an n-node graph, each feasible solution f consists of n edges, which can be specified as a permutation of n nodes denoted as π_f, where $\pi_f(0) = \pi_f(n)$, and $\pi_f(i) \neq \pi_f(j)$, $\forall i > 0, j > 0, i \neq j$. Assuming the starting node is j we have:

$$q_{\text{TSP}}(\pi_f \mid \pi_f(0) = j) = \prod_{i=1}^{n-1} q_{\text{TSP}}(\pi_f(i) \mid \pi_f(< i)). \tag{7}$$

To make $q_{\boldsymbol{\theta}}^{\text{TSP}}$ serve as an auxiliary distribution, we define:

$$q_{\text{TSP}}(\pi_f(i) \mid \pi_f(< i)) = \frac{\exp(\theta_{\pi_f(i-1), \pi_f(i)})}{\sum_{j=i}^{n} \exp(\theta_{\pi_f(i-1), \pi_f(j)})}. \tag{8}$$

Here the higher valued $\theta_{i,j}$ means the higher probability for the edge from node i to node j to be sampled. More importantly, notice that we use matrix $\boldsymbol{\theta} \in \mathbb{R}^{n \times n}$ to parameterize the probabilistic distribution of $n!$ discrete feasible solutions. The compact, continuous and differentiable space of θ allows us to leverage gradient-based optimization without costly MDP-based construction of feasible solutions, which has been the bottleneck for scaling up in representative DRL solvers so far. In other words, we also no longer need costly MCMC-based sampling for optimizing our model due to the chain-rule decomposition.

The last question is how to choose the first node in a tour. In this paper, we choose the starting node randomly with a uniform distribution, which yields:

$$q_{\boldsymbol{\theta}}^{\text{TSP}}(f) = \sum_{j=1}^{n} \frac{1}{n} \cdot q_{\text{TSP}}(\pi_f \mid \pi_f(0) = j). \tag{9}$$

3.2.2 Auxiliary Distribution for MIS

For the Maximal Independent Set (MIS) problem, the feasible solution is a set of independent nodes, which means that none of the node has any link to any other node in the same set. To ease the analysis, we further impose a constraint to the MIS solutions such that each set is not a proper subset of any other independent set in the feasible domain.

[2] TSP has a global constraint of forming a Hamiltonian cycle.

To enable the chain-rule decomposition in probability estimation, we introduce $\boldsymbol{a}$ as an ordering of the independent nodes in solution f, and $\{\boldsymbol{a}\}_f$ as the set of all possible orderings of the nodes in f. The chain rule applied to $\boldsymbol{a}$ can thus be defined as:

$$q_{\boldsymbol{\theta}}^{\text{MIS}}(f) = \sum_{\boldsymbol{a} \in \{\boldsymbol{a}\}_f} q_{\text{MIS}}(\boldsymbol{a}), \tag{10}$$

$$\begin{aligned} q_{\text{MIS}}(\boldsymbol{a}) &= \prod_{i=1}^{|\boldsymbol{a}|} q_{\text{MIS}}(a_i \mid \boldsymbol{a}_{<i}) \\ &= \prod_{i=1}^{|\boldsymbol{a}|} \frac{\exp(\theta_{a_i})}{\sum_{j \in \mathcal{G}(\boldsymbol{a}_{<i})} \exp(\theta_{a_j})}. \end{aligned} \tag{11}$$

where $\mathcal{G}(\boldsymbol{a}_{<i})$ denotes the set of available nodes for growing partial solution $(a_1, \ldots, a_{i-1})$, i.e., the nodes that are not connected to any nodes in $\{a_1, \ldots, a_{i-1}\}$. Notice again that the parameterization space for MIS $\boldsymbol{\theta} \in \mathbb{R}^n$ (where n denotes the number of nodes in the graph) is compact, continuous and differentiable, which allows efficient gradient-driven optimization.

Due to the space limit, we leave the proof of the convergence between $p_{\boldsymbol{\theta}}$ and $q_{\boldsymbol{\theta}}$ (i.e., $q_{\boldsymbol{\theta}}^{\text{TSP}}$ and $q_{\boldsymbol{\theta}}^{\text{MIS}}$) to the appendix.

3.3 Meta-Learning Framework

We train a graph neural network (GNN) over a collection of CO problems in a way that the it can capture the common nature across all the problems, and adapt its model parameters effectively to each problem instance based on the features/structure of each input graph. Especially, we follow the Medel-agnostic Mata Learning (MAML) framework [15] closely in the instance-specific initialization part of the optimization process. Let $F_{\boldsymbol{\Phi}}$ be the graph neural network with parameter $\boldsymbol{\Phi}$, and denote by $\boldsymbol{\kappa}_s$ the input features of an instance graph s in collection $\mathcal{C}$, by A_s the adjacency matrix of the input graph, and by $\boldsymbol{\theta}_s := F_{\boldsymbol{\Phi}}(\boldsymbol{\kappa}_s, A_s)$ the instance-specific initialization of model parameters. The vanilla loss function is defined as the expected cost of the solution for any graph in the collection as:

$$\mathcal{L}(\boldsymbol{\Phi} \mid \mathcal{C}) = \mathbb{E}_{s \in \mathcal{C}} \ell_q(\boldsymbol{\theta}_s) = \mathbb{E}_{s \in \mathcal{C}} \ell_q(F_{\boldsymbol{\Phi}}(\boldsymbol{\kappa}_s, A_s)). \tag{12}$$

The gradient-based updates can thus be written as:

$$\begin{aligned} \nabla_{\boldsymbol{\Phi}} \mathcal{L}(\boldsymbol{\Phi} \mid \mathcal{C}) &= \mathbb{E}_{s \in \mathcal{C}} \left[\nabla_{\boldsymbol{\Phi}} \boldsymbol{\theta}_s \cdot \nabla_{\boldsymbol{\theta}_s} \ell_q(\boldsymbol{\theta}_s) \right] \\ &= \mathbb{E}_{s \in \mathcal{C}} \left[\nabla_{\boldsymbol{\Phi}} F_{\boldsymbol{\Phi}}(\boldsymbol{\kappa}_s, A_s) \cdot \nabla_{\boldsymbol{\theta}_s} \ell_q(\boldsymbol{\theta}_s) \right]. \end{aligned} \tag{13}$$

where $\nabla_{\boldsymbol{\theta}_s} \ell_q(\boldsymbol{\theta}_s)$ is estimated using the REINFORCE algorithm (Equation 6). Since ℓ_q does not depend on the ground-truth labels, we can further fine-tune neural network parameters on each single test instance with REINFORCE-based updates, which is referred as *active search* [7, 22].

Specifically, the fine-tuned parameters $\boldsymbol{\Phi}_s^{(T)}$ is computed using one or more gradient updates for each graph instance s. For example, when adapting to a problem instance s using T gradient updates with learning rate α, we have:

$$\boldsymbol{\Phi}_s^{(0)} = \boldsymbol{\Phi}_s, \qquad \boldsymbol{\Phi}_s^{(t)} = \boldsymbol{\Phi}_s^{(t-1)} - \alpha \nabla_{\boldsymbol{\Phi}_s^{(t-1)}} \mathcal{L}(\boldsymbol{\Phi}_s^{(t-1)} \mid \{s\}) \quad \text{for} \quad 1 \leq t \leq T, \tag{14}$$

$$\boldsymbol{\theta}_s^{(T)} = F_{\boldsymbol{\Phi}_s^{(T)}}(\boldsymbol{\kappa}_s, A_s). \tag{15}$$

Next, we optimize the performance of the graph neural network with updated parameters (i.e., $\boldsymbol{\Phi}_s^{(T)}$) with respect to $\boldsymbol{\Phi}$, with a meta objective:

$$\mathcal{L}_{\text{meta}}(\boldsymbol{\Phi} \mid \mathcal{C}) = \mathbb{E}_{s \in \mathcal{C}} \ell_q(\boldsymbol{\theta}_s^{(T)}), \tag{16}$$

and calculate the meta-updates as:

$$\begin{aligned} \nabla_{\boldsymbol{\Phi}} \mathcal{L}_{\text{meta}}(\boldsymbol{\Phi} \mid \mathcal{C}) &= \mathbb{E}_{s \in \mathcal{C}} \left[\nabla_{\boldsymbol{\Phi}} \boldsymbol{\theta}_s^{(T)} \cdot \nabla_{\boldsymbol{\theta}_s^{(T)}} \ell_q(\boldsymbol{\theta}_s^{(T)}) \right] \\ &\approx \mathbb{E}_{s \in \mathcal{C}} \left[\nabla_{\boldsymbol{\Phi}_s^{(T)}} F_{\boldsymbol{\Phi}_s^{(T)}}(\boldsymbol{\kappa}_s, A_s) \cdot \nabla_{\boldsymbol{\theta}_s^{(T)}} \ell_q(\boldsymbol{\theta}_s^{(T)}) \right]. \end{aligned} \tag{17}$$

Notice that we adopt the first-order approximation to optimize this objective, which ignores the update via the gradient term of $\nabla_{\boldsymbol{\Phi}} \mathcal{L}(\boldsymbol{\Phi} \mid \{s\})$. We defer the derivation of the approximation formula to the appendix. Alg. 1 in the appendix illustrates the full training process of the meta-learning.

3.4 Per-Instance Search

In this section, we describe the decoding strategies used in our paper. Given a fine-tuned continuous parameterization (i.e., heatmap) of the solution space $\boldsymbol{\theta}_s^{(T)}$, the decoding strategy aims to search a feasible solution that minimizes the cost function c.

Greedy Decoding generates the solution by a sequential decoding process similar to the auxiliary distribution designed for each combinatorial optimization problem, where at each step, the variable k with the highest score θ_k is chosen to extend the partial solution. For TSP, the first node in the permutation is randomly picked.

Sampling Inspired by Kool et al. [34], we propose to parallelly sample multiple solutions according the auxiliary distribution, and report the best one. The parallel sampling of solutions in DIMES are very efficient due to the fact that it only relies on the final heatmap $\boldsymbol{\theta}_s^{(T)}$ but not on neural networks.

Monte Carlo Tree Search Inspired by [16], for the TSP task, we also leverage a more advanced reinforcement learning-based searching approach, i.e., Monte Carlo tree search (MCTS), to find high-quality solutions. In MCTS, k-opt transformation actions are sampled guided by the continuous parameterization $\boldsymbol{\theta}_s^{(T)}$ to improve the current solutions. The MCTS iterates over the simulation, selection, and back-propagation steps, until no improving actions exists among the sampling pool. For more details, please refer to [16].

3.5 Graph Neural Networks

Based on the shape of the differentiable variable $\boldsymbol{\theta}$ required by each problem (i.e., $\mathbb{R}^{n\times n}$ for TSP and $\mathbb{R}^n$ for MIS), we use Anisotropic Graph Neural Networks [9] and Graph Convolutional Networks [32] as the backbone network for TSP and MIS tasks, respectively. Due to the space limit, the detailed neural architecture design can be found in the appendix.

4 Experiments

4.1 Experiments for Traveling Salesman Problem

4.1.1 Experimental Settings

Data Sets the training instances are generated on the fly. We closely follow the data generation procedure of previous works, e.g., [35]. We generate 2-D Euclidean TSP instances by sampling each node independently from a uniform distribution over the unit square. The TSP problems of different scales are named TSP-500/1000/10000, respectively, where TSP-N indicates the TSP instance on N nodes. For testing, we use the test instances generated by Fu et al. [16]. There are 128 test instances in each of TSP-500/1000, and 16 test instances in TSP-10000.

Evaluation Metrics Evaluation Metrics: for model comparison, we report the average length (Length), average performance drop (Drop) and averaged inference latency time (Time), respectively, where *Length* (the shorter, the better) is the average length of the system-predicted tour for each test-set graph, *Drop* (the smaller, the better) is the average of relative performance drop in terms of the solution length compared to a baseline method, and *Time* (the smaller, the better) is the clock time for generating a solution for a single test instance in seconds (s), minutes (m), or hours (h).

Training and Hardware Due to the space limit, please refer to the appendix.

4.1.2 Main Results

Our main results are summarized in Table 1, with $T = 15$ for TSP-500, $T = 14$ for TSP-1000, and $T = 12$ for TSP-10000. We compare DIMES with 14 other TSP solvers on the same test sets. We divide those 14 methods into two categories: 6 traditional OR methods and 8 learning-based methods.

Table 1: Results on TSP-500/1000/10000. See Section 4.1.2 for detailed descriptions. * indicates the baseline for computing the performance drop. Results of baselines except the running time of Att-GCN are taken from Fu et al. [16].

Method	Type	TSP-500			TSP-1000			TSP-10000		
		Length ↓	Drop ↓	Time ↓	Length ↓	Drop ↓	Time ↓	Length ↓	Drop ↓	Time ↓
Concorde	OR (exact)	16.55*	—	37.66m	23.12*	—	6.65h	N/A	N/A	N/A
Gurobi	OR (exact)	16.55	0.00%	45.63h	N/A	N/A	N/A	N/A	N/A	N/A
LKH-3	OR	16.55	0.00%	11.41m	23.12	0.00%	38.09m	71.77*	—	8.8h
Nearest Insertion	OR	20.62	24.59%	0s	28.96	25.26%	0s	90.51	26.11%	6s
Random Insertion	OR	18.57	12.21%	0s	26.12	12.98%	0s	81.85	14.04%	4s
Farthest Insertion	OR	18.30	10.57%	0s	25.72	11.25%	0s	80.59	12.29%	6s
EAN	RL+S	28.63	73.03%	20.18m	50.30	117.59%	37.07m	N/A	N/A	N/A
EAN	RL+S+2-OPT	23.75	43.57%	57.76m	47.73	106.46%	5.39h	N/A	N/A	N/A
AM	RL+S	22.64	36.84%	15.64m	42.80	85.15%	63.97m	431.58	501.27%	12.63m
AM	RL+G	20.02	20.99%	1.51m	31.15	34.75%	3.18m	141.68	97.39%	5.99m
AM	RL+BS	19.53	18.03%	21.99m	29.90	29.23%	1.64h	129.40	80.28%	1.81h
GCN	SL+G	29.72	79.61%	6.67m	48.62	110.29%	28.52m	N/A	N/A	N/A
GCN	SL+BS	30.37	83.55%	38.02m	51.26	121.73%	51.67m	N/A	N/A	N/A
Att-GCN	SL+MCTS	16.97	2.54%	5.91m	23.86	3.22%	12.47m	74.93	4.39%	16.73m
DIMES (ours)	RL+G	17.81	7.61%	1.23m	24.91	7.74%	2.21m	80.45	12.09%	2.64m
DIMES (ours)	RL+S	17.80	7.55%	1.24m	24.89	7.70%	2.66m	80.42	12.05%	2.86m
DIMES (ours)	RL+MCTS	**16.84**	**1.76%**	2.98m	**23.69**	**2.46%**	4.47m	**74.06**	**3.19%**	19.97m

Table 2: Ablation study on TSP-1000.

(a) On meta-learning ($T = 10$).

Meta-update	Fine-tuning	Length ↓
		27.11
✓		26.58
	✓	25.68
✓	✓	**25.09**

(b) On fine-tuning parts ($T = 5$).

Part	Length ↓
Heatmap	27.73
MLP	26.75
GNNOut+MLP	**26.49**
GNN+MLP	26.81

(c) On meta-update steps T.

T	0	4	8	10	12	14
Length ↓	25.79	25.28	25.08	25.08	24.97	24.91

(d) On heatmaps for MCTS.

Heatmap	Length ↓
$\mathrm{Unif}(0, 1)$	25.52
$1/(r_i + 1)$	24.14
Att-GCN	23.86
DIMES (ours)	**23.69**

- The traditional operations research methods include two exact solvers, i.e., Concorde [4] and Gurobi [18], and a strong heuristic solver named LKH-3 [19]. We also compare against simple heuristics including Nearest, Random, and Farthest Insertion.

- The learning based methods include 8 variants of the 4 methods with the strongest results in recent benchmark evaluations, namely EAN [13], AM [35], GCN [24], and Att-GCN [16], respectively. Those methods can be further divided into the reinforcement learning (RL) sub-category and the supervised learning (SL) sub-category. Results of baselines except the running time of Att-GCN are taken from Fu et al. [16]. We re-run the publicly available code of Att-GCN on our hardware to ensure fair comparison of time.

The decoding schemes in each method (if applicable) are further specified as Greedy decoding (G), Sampling (S), Beam Search (BS), and Monte Carlo Tree Search (MCTS). The 2-OPT improvements [11] can be optionally used to further improve the neural network-generated solution via heuristic local search. See Section 3.4 for a more detailed descriptions of the various decoding techniques.

As shown in the table, DIMES significantly outperforms many previous learning-based methods. Notably, although DIMES is trained without any ground truth solutions, it is able to outperform several supervised methods. DIMES also consistently outperforms simple traditional heuristics. The best performance is achieved by RL+MCTS, which requires considerably more time. RL+G/S are faster than RL+MCTS and are competitive to the simple heuristics. Moreover, they are still better than many previous learning-based methods in terms of solution quality and inference time.

Table 3: Results of various methods on MIS problems. Notice that we disable graph reduction and 2-opt local search in all models for a fair comparison, since it is pointed out by [8] that all models would perform similarly with a local search post-processing. See Section 4.2.2 for detailed descriptions. * indicates the baseline for computing the performance drop.

Method	Type	SATLIB			ER-[700-800]			ER-[9000-11000]		
		Size ↑	Drop ↓	Time ↓	Size ↑	Drop ↓	Time ↓	Size ↑	Drop ↓	Time ↓
KaMIS	OR	425.96*	—	37.58m	44.87*	—	52.13m	381.31*	—	7.6h
Gurobi	OR	425.95	0.00%	26.00m	41.38	7.78%	50.00m	N/A	N/A	N/A
Intel	SL+TS	N/A	N/A	N/A	38.80	13.43%	20.00m	N/A	N/A	N/A
Intel	SL+G	420.66	1.48%	23.05m	34.86	22.31%	6.06m	284.63	25.35%	5.02m
DGL	SL+TS	N/A	N/A	N/A	37.26	16.96%	22.71m	N/A	N/A	N/A
LwD	RL+G	422.22	0.88%	18.83m	41.17	8.25%	6.33m	**345.88**	**9.29%**	7.56m
DIMES (ours)	RL+G	421.24	1.11%	24.17m	38.24	14.78%	6.12m	320.50	15.95%	5.21m
DIMES (ours)	RL+S	**423.28**	**0.63%**	20.26m	**42.06**	**6.26%**	12.01m	332.80	12.72%	12.51m

4.1.3 Ablation Study

On Meta-Learning To study the efficacy of meta-learning, we consider two dimensions of ablations: (i) with or without meta-updates: whether to use $f_{\boldsymbol{\Phi}}(\kappa_s, A_s)$ or $\boldsymbol{\theta}_s^{(T)}$ in the objective function; (ii) with or without fine-tuning in the inference phase. The results on TSP-1000 with $T = 10$ and greedy decoding are summarized in Table 2a. Both meta-updates and fine-tuning are crucial to the performance of our method. That is because meta-learning helps the model generalize across problem instances, and fine-tuning helps the trained model adapt to each specific problem instance.

On Fine-Tuning Parts We study the effect of fine-tuning parts during both training and testing. In general, the neural architecture we used is a GNN appended with a MLP, whose output is called heatmaps. We consider the following fine-tuning parts: (i) the heatmap; (ii) the parameter of MLP; (iii) the output of GNN and the parameters of MLP; (iv) the parameters of GNN and MLP. Table 2b summarizes the results with various fine-tuning parts for TSP-1000 with $T = 5$ and greedy decoding. The result demonstrates that (iii) works best. We conjecture that (iii) makes a nice trade-off between universality and variance reduction.

On Meta-Update Steps We also study the effect of the number T of meta-update steps during training. Table 2c shows the test performance on TSP-1000 by greedy decoding with various T's. As the number of meta-updates increases, the test performance improves accordingly. Meanwhile, more meta-update steps consumes more training time. Hence, there is a trade-off between performance and training time in practice.

On Heatmaps for MCTS To study where the DIMES heatmap is essential to the good performance in MCTS, we replace it with the following heatmaps: (i) each value is independently sampled from $\mathrm{Unif}(0, 1)$; (ii) $1/(r_i + 1)$, where $r_i \geq 1$ denotes the rank of the length of the i-th edge among those edges that share the source node with it. This can be regarded as an approximation to the nearest neighbor heuristics. We also compare with the Att-GCN heatmap [16]. The comparison of heatmaps for TSP-1000 by MCTS is shown in Table 2d. The result confirms that the DIMES heatmap does not simply learn nearest neighbor heuristics, but can identify non-trivial good candidate edges.

4.2 Experiments For Maximal Independent Set

4.2.1 Experimental Settings

Data Sets We mainly focus on two-types of graphs that recent work [41, 1, 8] shows struggles against, i.e., Erdős-Rényi (ER) graphs [14] and SATLIB [20], where the latter is a set of graphs reduced from SAT instances in CNF. The ER graphs of different scales are named ER-[700-800] and ER-[9000-11000], where ER-[N-M] indicates the graph contains N to M nodes. The pairwise connection probability p is set to 0.15 and 0.02 for ER-[700-800] and ER-[9000-11000], respectively. The 4,096 training and 5,00 test ER graphs are randomly generated. For SATLIB, which consists of 40,000 instances, of which we train on 39,500 and test on 500. Each SAT instance has between

403 to 449 clauses. Since we cannot find the standard train-test splits for both SAT and ER graphs datasets, we randomly split the datasets and re-run all the baseline methods.

Evaluation Metrics To compare the solving ability of various methods, we report the average size of independent set (Size), average performance drop (Drop) and latency time (Time), respectively, where *Size* (the larger, the better) is the average size of the system-predicted maximal independent set for each test-set graph, *Drop* and *Time* are defined similarly as in Sec. 4.1.1.

Training and Hardware Due to the space limit, please refer to the appendix.

4.2.2 Main Results

Our main results are summarized in Table 3, where our method (last line) is compared 6 other MIS solvers on the same test sets, including two traditional OR methods (i.e., Gurobi and KaMIS) and four learning-based methods. For Gurobi, we formulate the MIS problem as a integer linear program. For KaMIS, we use the code unmodified from the official repository[3]. The four learning based methods can be divided into the reinforcement learning (RL) category, i.e., S2V-DQN [29] and LwD [1]) and the supervised learning (SL) category, i.e., Intel [41] and DGL [8].

We produced the results for all the learning based methods by running an integrated implementation[4] provided by Böther et al. [8]. Notice that as pointed out by Böther et al. [8], the graph reduction and local 2-opt search [3] can find near-optimal solutions even starting from a randomly generated solution, so we disable the local search or graph reduction techniques during the evaluation for all learning based methods to show their real CO solving ability. The methods that cannot produce results in $10\times$ time limit of DIMES are labeled as N/A.

As shown in Table 3, our DIMES model outperforms the previous baseline methods on the medium-scale SATLIB and ER-[700-800] datasets, and significantly outperforms the supervised baseline (i.e., Intel) on the large-scale ER-[9000-11000] setting. This shows that supervised neural CO solvers cannot well solve large-scale CO problem due to the expensive annotation problem and generalization problem. In contrast, reinforcement-learning methods are a better choice for large-scale CO problems. We also find that LwD outperforms DIMES on the large-scale ER-[9000-11000] setting. We believe this is because LwD is specially designed for locally decomposable CO problems such as MIS and thus can use parallel prediction, but DIMES are designed for general CO problems and only uses autoregressive factorization. How to better utilize the fact of local decomposability of MIS-like problems is one of our future work.

5 Concluding Remarks

Scalability without significantly scarifying approximation accuracy is a critical challenge in combinatorial optimization. In this work we proposed DIMES, a differentiable meta solver that is able to solve large-scale combinatorial optimization problems effectively and efficiently, including TSP and MIS. The novel parts of DIMES include the compact continuous parameterization and the meta-learning strategy. Notably, although our method is trained without any ground truth solutions, it is able to outperform several supervised methods. In comparison with other strong DRL solvers on TSP and MIS problems, DIMES can scale up to graphs with over ten thousands nodes while the others either fail to scale up, or can only produce significantly worse solutions instead in most cases.

One limitation of DIMES is that the heatmap $\boldsymbol{\theta}$ is generated in one-shot without intermediate steps, which could potentially limit the reasoning power of our method, as shown in the MIS task. Another limitation is that applying DIMES to a broader ranges of NP-complete problems that variables can take multiple values, such as Mixed Integer Programming (MIP), is not trivial and needs further understanding of the nature of the problems.

[3] `https://github.com/KarlsruheMIS/KaMIS` (MIT License)

[4] `https://github.com/MaxiBoether/mis-benchmark-framework` (No License)

References

[1] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *International Conference on Machine Learning*, pages 134–144. PMLR, 2020.

[2] Ismail R Alkhouri, George K Atia, and Alvaro Velasquez. A differentiable approach to combinatorial optimization using dataless neural networks. *arXiv preprint arXiv:2203.08209*, 2022.

[3] Diogo V Andrade, Mauricio GC Resende, and Renato F Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547, 2012.

[4] David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde TSP solver. `https://www.math.uwaterloo.ca/tsp/concorde/index.html`, 2006.

[5] Sanjeev Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 2–11. IEEE, 1996.

[6] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962.

[7] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[8] Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=mk0HzdqY7i1`.

[9] Xavier Bresson and Thomas Laurent. An experimental study of neural networks for variable graphs. 2018.

[10] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 6281–6292, 2019.

[11] Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6 (6):791–812, 1958.

[12] Paulo R de O da Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. *arXiv preprint arXiv:2004.01608*, 2020.

[13] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the TSP by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 170–181. Springer, 2018.

[14] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[15] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[16] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large TSP instances. *arXiv preprint arXiv:2012.10658*, 2020.

[17] Teofilo F Gonzalez. *Handbook of approximation algorithms and metaheuristics*. CRC Press, 2007.

[18] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018.

[19] K. Helsgaun. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. Technical report, Roskilde University, 2017.

[20] Holger H Hoos and Thomas Stützle. Satlib: An online resource for research on sat. *Sat*, 2000: 283–292, 2000.

[21] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. *arXiv preprint arXiv:1911.09539*, 2019.

[22] André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems. *arXiv preprint arXiv:2106.05126*, 2021.

[23] Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph neural network guided local search for the traveling salesperson problem. *arXiv preprint arXiv:2110.05291*, 2021.

[24] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

[25] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. On learning paradigms for the travelling salesman problem. *arXiv preprint arXiv:1910.07210*, 2019.

[26] Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, Thomas Laurent, and Xavier Bresson. Learning tsp requires rethinking generalization. *arXiv preprint arXiv:2006.07054*, 2020.

[27] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33:6659–6672, 2020.

[28] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[29] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[30] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 30: 6348–6358, 2017.

[31] Minsu Kim, Jinkyoo Park, et al. Learning collaborative policies to solve np-hard routing problems. *Advances in Neural Information Processing Systems*, 34, 2021.

[32] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[33] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[34] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[35] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

[36] Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! 2019.

[37] Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. Deep policy dynamic programming for vehicle routing problems. *arXiv preprint arXiv:2102.11756*, 2021.

[38] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *arXiv preprint arXiv:2010.16011*, 2020.

[39] Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 34, 2021.

[40] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[41] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.

[42] Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*, 2020.

[43] Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv preprint arXiv:1911.04936*, 2019.

[44] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems*, 34, 2021.

[45] John W Moon and Leo Moser. On cliques in graphs. *Israel journal of Mathematics*, 3(1):23–28, 1965.

[46] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V Snyder, and Martin Takáč. Reinforcement learning for solving the vehicle routing problem. *arXiv preprint arXiv:1802.04240*, 2018.

[47] Wenbin Ouyang, Yisen Wang, Shaochen Han, Zhejian Jin, and Paul Weng. Improving generalization of deep reinforcement learning-based tsp solvers. *arXiv preprint arXiv:2110.02843*, 2021.

[48] Christos H Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical computer science*, 4(3):237–244, 1977.

[49] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[50] Bo Peng, Jiahai Wang, and Zizhen Zhang. A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In *International Symposium on Intelligence Computation and Applications*, pages 636–650. Springer, 2019.

[51] Colin Reeves. Modern heuristics techniques for combinatorial problems. *Nikkan Kogyo Shimbun*, 1997.

[52] Paul Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 46, 1997.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[54] Natalia Vesselinova, Rebecca Steinert, Daniel F Perez-Ramirez, and Magnus Boman. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 8:120388–120416, 2020.

[55] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in Neural Information Processing Systems*, 28:2692–2700, 2015.

[56] Chenguang Wang, Yaodong Yang, Oliver Slumbers, Congying Han, Tiande Guo, Haifeng Zhang, and Jun Wang. A game-theoretic approach for improving generalization ability of tsp solvers. *arXiv preprint arXiv:2110.15105*, 2021.

[57] Runzhong Wang, Zhigang Hua, Gan Liu, Jiayi Zhang, Junchi Yan, Feng Qi, Shuang Yang, Jun Zhou, and Xiaokang Yang. A bi-level framework for learning to solve combinatorial optimization on graphs. *arXiv preprint arXiv:2106.04927*, 2021.

[58] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[59] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems.. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[60] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *Proceedings of 35th AAAI Conference on Artificial Intelligence*, 2021.

[61] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems*, 34, 2021.

[62] Emre Yolcu and Barnabás Póczos. Learning local search heuristics for boolean satisfiability. In *NeurIPS*, pages 7990–8001, 2019.

[63] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. *arXiv preprint arXiv:2010.12367*, 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 Sect. 5.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] No obvious negative social impact.
   (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? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
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)? [Yes] See Sect. C in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Sect. C in the supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] No previous work reports error bars.
   (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 Sect. C in the supplementary material.
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]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] No new assets.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] They are publicly available.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No such 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]
   (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]