Learning Heuristics for Minimum Latency Problem with RL and GNN

Siqi Hao **Salar Hosseini Khorasgani** **Philip Huang** **Mohamed Khodeir**

Abstract

Recent work has successfully applied methods from reinforcement learning (RL) to derive domain-dependent heuristics for challenging combinatorial optimization problems from a set of training instances. This approach is promising because it presents an opportunity to learn heuristics which are tailored to the patterns/regularities in a specific distribution of instances that we care to solve, without the need for dedicated engineering efforts. We build on this idea and show how RL can be applied to the Minimum Latency Problem by using a graph attention network to encode a stochastic policy for constructively building partial paths, yielding solutions which are comparable to state-of-the-art, hand-engineered methods. We provide an extensive empirical evaluation of the RL approach, situating its performance characteristics in the context of existing methods. With iterative local search, we can further improve the quality of solutions constructed form RL to acheive excellent performance-runtime tradeoff, even on large instances. Our code is available at `https://github.com/philip-huang/mie1666_project`.

1 Introduction

Traveling salesman problem (TSP) tries to find the shortest Hamiltonian cycle that visits every node in a weighted and complete graph. However, the TSP problem is not oriented for customers, who might wish to spend less time waiting for a product delivery. Minimal latency problem (also known as traveling repairman/deliveryman) tries to minimize the total latency at each node. The latency of a node is defined as the wait time before that node is visited by the traveling repairmen. Mathematically, given a graph $G = (V, E)$ with a distance matrix that defines the cost $c(v_1, v_2)$ between every pair of nodes, let $\pi = \{\pi_0, \pi_1, \pi_2, ..., \pi_n\}$ be an ordered permutation of all the nodes (i.e. a Hamiltonian path) and assume that π_0 is a fixed starting node for the problem instance. The objective in TSP is to minimize

$$\min_{\pi} \sum_{i=0}^{n-1} c(\pi_i, \pi_{i+1}) + c(\pi_n, \pi_0) \tag{1}$$

The objective of the minimal latency problem (MLP) is

$$\min_{\pi} \sum_{i=1}^{n} \sum_{j=0}^{i-1} c(\pi_j, \pi_{j+1}) \tag{2}$$

The minimum latency problem arises in many applications including job scheduling, when the order in which jobs are scheduled can affect the total completion time [1]. While it may seem at first glance that the optimal solution to a minimum latency problem can be simply obtained from solving TSP, we illustrate the differences between them with a simple example in Figure 1.

Since the graph is 1-d, solving the optimal TSP problem is easy. For example, the route $\pi = [s, t_1, t_2, t_3, t_4]$ is a minimum TSP loop with a total latency of $1 + 7 + 16 + 27 = 51$. However, the path with minimal latency route is actually $\pi^* = [s, t_3, t_1, t_2, t_4]$, which has a total latency of $2 + 5 + 11 + 31 = 49$. Hence, finding the minimal latency path is different from solving TSP and is non trivial even in 1-d.

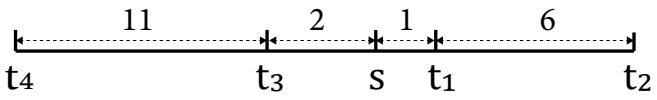

Figure 1: A 1-d line graph with five vertex on the axis. s is the starting location. The number denotes the cost to travel between two adjacent vertices. A distance matrix between every pair node can be easily constructed to make a complete graph. For example, the distance between t_2 and t_4 is 20

Our primary contribution is to apply reinforcement learning and attention-based graph neural networks to solve the minimum latency problem. While these graph-learning-based algorithmic frameworks have been studied in many graph optimization problems already (i.e. Traveling Salesman, Vehicle Routing Problem, Maximum Cut), we demonstrate the ability of the framework to generalize to this new NP-hard graph optimization problems and show that our solutions are on par with domain-specific, hand-engineered solutions from literature.

2 Related Work

Minimum latency problem is first studied in the literature in [2] as traveling repairman problem and later formulated as MLP in [3]. The problem is also known as the delivery man problem [4]. Since then, many have attempted to find exact and approximate algorithms for MLP. One family of approaches formulates MLP as a mixed integer programming problem [1, 5, 6] and uses existing solvers (e.g. CPLEX) to solve them. In [7], the authors use a branch-cut-and price algorithm to find optimal solutions for graphs with up to 106 nodes, the largest graph in MLP literature with a known optimal solution.

One of the earliest approximation algorithms in MLP is given by Blum et.al. [3], which reports a polynomial time, 72-approximation ratio algorithm. Chaudhuri et. al. [8] gives a 3.59-approximation algorithm, which tries to find a path by stitching smaller sub-paths. Meta-heuristic is another class of algorithm that combines different heuristics to solve problems. A state-of-the-art meta-heuristic, GILS-RVND [9], can produce high-quality solution for up to 1000 nodes. Recently, Santana [10] builds on GILS-RVND with data mining to improve solution quality and reduce computational time.

Many classical methods for the combinatorial optimization (CO) problems use hand-crafted heuristics to construct the solutions, which can be suboptimal. Recently, reinforcement learning (RL) algorithms [11, 12, 13] are applied to this field to automate the learning of the heuristics by training an agent to construct the solution in the subsequential order. Graph neural networks (GNNs) play an important role in learning the representation of the graph instances. GNNs are invariant and equivariant to the permutations of the nodes by design. By aggregating the information of the neighboring nodes, GNNs can naturally learn the graph encodings that are helpful in the context of CO [14].

Dai et al. [15] use a Structure2Vec (S2V) GNN to encode the nodes of the graph instances. The authors combine the Deep Q-learning (DQN) with S2V to learn the node selection heuristic for TSP that inserts the nodes to the partial solution iteratively. Kool et al. [16] propose an encoder-decoder architecture to solve TSP and several variants. The attention-based encoder and decoder allow message passing between nodes, and adjust the model for different variants by utilizing special features, e.g., the demands and penalties for each node. The authors train the model using REINFORCE with a rollout baseline. In this project, we apply RL and a GNN to the minimum latency problem, which has not been solved by ML approaches in previous works.

3 Methodology

To solve the minimum latency problem, we apply the attention-based model by Kool et al. [16] which can parameterize a heuristic (policy) for selecting nodes to visit in a constructive manner. The proposed method is suitable for tackling this problem because its ability to learn heuristics has already been demonstrated on multiple routing problems (e.g. TSP, Vehicle Routing Problem,

and Orienteering Problem) of reasonable size (graphs with up to 100 nodes) with a single set of hyperparameters [16]. Moreover, by using an autoregressive encoder-decoder model to encode a policy, the outputs can be conditioned on partial paths, which has been shown to be more effective than some previous non-autoregressive approaches [16].

Given an MLP problem instance s with one fixed starting node and n nodes to visit, a path $\pi = \{\pi_0, \pi_1, \pi_2, ..., \pi_n\}$ (Hamiltonian path through all graph nodes with π_0 fixed) is constructed using a stochastic policy $p_\theta(\pi|s) = \prod_{t=1}^{n} p_\theta(\pi_t|s, \pi_{0:t-1})$ with parameters θ. To encode this policy, a Graph Attention Network [17] is used which consists of an encoder and a decoder. These components and the training scheme are described in the subsections below. An overview of the Graph Attention Network used to encode the node selection policy is shown in Figure 2.

3.1 Encoder and decoder

The encoder takes as input the features $\mathbf{x}_i$ ($i \in \{0, ..., n\}$) of the $n + 1$ nodes in the graph. In our case, we use the 2d coordinates of each node as its input feature when handling symmetric graphs (no service time at each node), or the 2d coordinates concatenated with the service time when handling asymmetric graphs (can have non-zero service times at each of the n nodes to visit, as further discussed in Section 4.1). Using an initial linear projection followed by a sequence of N attention layers [18], the encoder outputs an embedding $\mathbf{h}_i^{(N)}$ for each node, as well as an aggregated embedding $\mathbf{h}_{(g)}^{(N)}$ for the graph from averaging all node embeddings. The dimension of the node embeddings is constant throughout the attention layers and is defined by a hyperparameter d_h. The embeddings are projected to this dimension d_h by the initial linear projection. Similar to [16], each of the N attention layers consists of (i) a multi-head attention (MHA) sublayer with M attention heads and (ii) a fully connected sublayer with hidden dimension d_{FC} and ReLU activation. Each sublayer uses batch normalization and has a skip connection with the previous sublayer.

The $n + 1$ node embeddings $\mathbf{h}_i^{(N)}$ and the graph embedding $\mathbf{h}^{(N)}$ are then used as inputs to the decoder, which sequentially computes for each time step t, the probability of the next node being node i given the partially constructed path, $p_\theta(\pi_t = i|s, \pi_{0:t-1})$. The node selection at each time step is done using either the greedy decoding method which selects the node with the highest probability, or sampling-based decoding which randomly samples N_{sample} solutions from the output distribution of the model and chooses the best. To keep track of the decoder state over the n time steps, a context embedding $\mathbf{h}_{(c)}^{(N)}$ which consists of the graph embedding $\mathbf{h}_{(g)}^{(N)}$ concatenated with the embedding of the last node selected $\mathbf{h}_{t-1}^{(N)}$, is passed through a multi-headed attention layer [18] at each time step to compute a new context node embedding $\mathbf{h}_{(c)}^{(N+1)}$. Then, a single attention head and a softmax layer are used to compute the probability of selecting each node. For the MLP problem, the embedding of the starting node $\mathbf{h}_0^{(N)}$ is used in the context at $t = 1$ but does not need to be kept in the context for later time steps because it will not be revisited at the end of the path. Additionally, whenever a node is selected, it is masked out to prevent further selection.

3.2 Policy training

To train the policy, the loss function is defined as:

$$\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[L(\pi)] \tag{3}$$

where $L(\pi)$ is the MLP path cost being minimized in Equation 2. In an RL context, this would correspond to having a discount factor of 1 with the total reward being the negative of the MLP path cost. This loss is optimized by gradient descent using the REINFORCE [19] gradient estimate:

$$\nabla\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[(L(\pi) - b(s))\nabla \log p_\theta(\pi|s)] \tag{4}$$

where $b(s)$ is a baseline path cost that the model is trained to improve over. The baseline term reduces gradient variance and increases learning speed while not biasing the gradient [19]. The baseline used here is the greedy rollout baseline [16] which is the cost of a solution from a greedy decoding of the best policy so far. The baseline policy is compared with the current training policy at the end of every epoch and is replaced by the training policy if the improvement is significant according to a paired t-test with $\alpha = 0.05$ over 10000 evaluation instances.

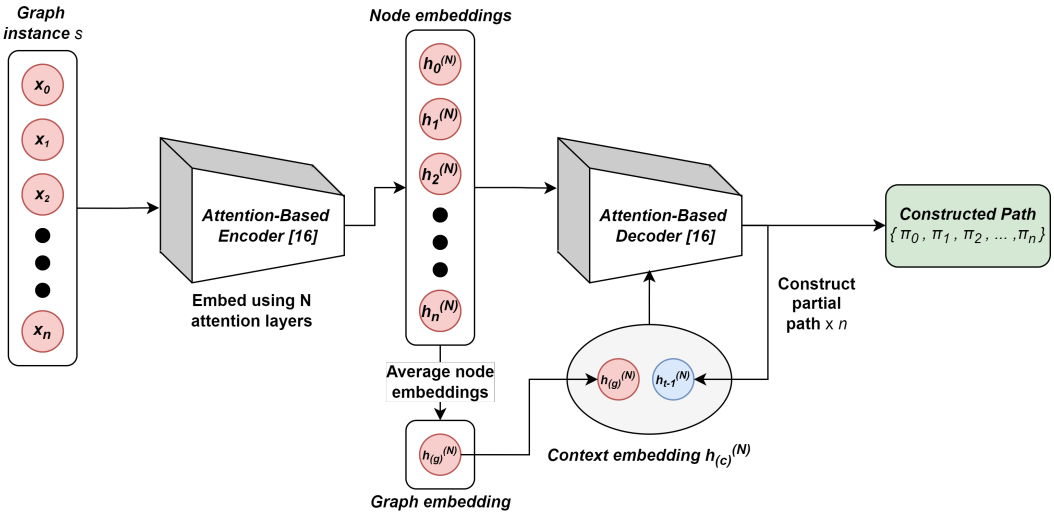

Figure 2: The graph attention network used to encode the node selection policy. First, the attention-based encoder produces node embeddings $\mathbf{h}_i^{(N)}$ and an aggregated graph embedding $\mathbf{h}_{(g)}^{(N)}$. Then, the node embeddings and a context embedding $\mathbf{h}_{(c)}^{(N)}$ (concatenation of $\mathbf{h}_{(g)}^{(N)}$ and the embedding of the last node selected $\mathbf{h}_{t-1}^{(N)}$) are inputted to the attention-based decoder to compute the probabilities of selecting each node. In the first time step, $\mathbf{h}_{t-1}^{(N)}$ is simply $\mathbf{h}_0^{(N)}$, the embedding of the fixed starting node. A node is selected for addition to the partial path using either greedy or sampling-based decoding, and it is masked out to prevent further selection.

3.3 Combining RL with Iterative Local Search (ILS)

Since solving MLP with RL is similar to running any construction-based heuristic method, we can apply local search methods directly to the output path from RL. Given a greedily decoded RL solution, we run iterative local search until the solution cannot be further improved locally. This is equivalent to running only one iteration of our meta-heuristic baseline, GILS-RVND, with greedy-decoding RL as the construction heuristic. Each step of the local search consists of the following: 1) search for a better neighboring solution and 2) make a random perturbation. Overall, this approach can improve the quality of the RL solution while maintaining low run time even for a large number of nodes.

3.4 Hyperparameter specification

Following [16], we set the number of attention layers to $N = 3$, the node embedding dimension to $d_h = 128$, number of attention heads in each MHA layer to $M = 8$, the hidden dimension of the fully connected sublayer in each attention layer to $d_{FC} = 512$, and the number of solutions to sample for sampling-based decoding to $N_{\text{sample}} = 1280$. We use a constant learning rate of 10^{-4}. We did not find it necessary to alter these hyperparameters since [16] already demonstrated these parameters to generalize to multiple routing problems, and as shown later in Section 5, we observed a steady decrease in the training loss and no overfitting issues.

4 Experimental Setup

4.1 Dataset

Similar to many related works [15, 16] that use RL for TSP, there are two main sources for training and test data. First, we can generate synthetic datasets by sampling random locations in 2-D and creating a distance matrix with a distance function (i.e. Euclidean distance). Extending data generation from a 2-D Euclidean plane to higher dimensions/other distance functions is straightforward. We can control the difficulty of the synthetic data easily by choosing the number of locations. Also, we can make the distance matrix be asymmetric by adding different service times at each node to the cost of every

outgoing edge from that node. Since there is no barrier in constructing synthetic data, we believe it can serve as a great source of training and test data.

TSPLib [20] is another popular real world benchmark for the traveling salesman problem, and every instance can be treated as a minimum latency problem. However, TSPLib has a significant drawback. Solving MLP is much harder than TSP, and there are only a few instances (< 10) that have less than 30 nodes. Since it is prohibitive to find optimal solutions for MLP for larger graphs, we will primarily use synthetic data for training and testing.

Following prior work [1], we generate 3 different classes of graphs for training and evaluation. For all 3 classes, point locations are sampled uniformly from a 2D plane, and the cost matrix which defines the edge weights has its elements $c_{ij} = t_{ij} + s_i$, where t_{ij} is the euclidean distance between nodes i and j and s_i is the *service time* for node i. The three classes S0, S1, and S2 differ only in how s_i is defined. Instances in S0 have $s_i = 0$, which means that S0 consists of symmetric graphs. S1 represents the "deliveryman" problem, so that s_i is low compared to t_{ij} - for these, s_i is sampled uniformly from $[0, \frac{t_{max}-t_{min}}{2}]$. Finally S2 instances are meant to represent the "repairman" problem, where service times are high, so s_i is sampled from $[\frac{t_{max}+t_{min}}{2}, \frac{3t_{max}-t_{min}}{2}]$. The different characteristics inherent in these three classes of graphs provide an interesting way to evaluate the ability of the RL approach to generalize to out of distribution problems.

4.2 Baseline Methods

Exact: CPLEX MIP. We follow prior work [1] and implement the MLP as an integer program using CPLEX. The formulation we adopt uses $O(n^2)$ binary variables, $O(n^3)$ real variables, and $O(n^2)$ constraints. We only use this baseline to evaluate our approach on relatively small graphs with up to 25 nodes, since solving larger instances to optimality is prohibitively expensive. We use default CPLEX settings, and run each test with a 2 hour time limit.

Heuristic: Nearest Neighbor (NN). A very simple construction heuristic for the MLP is to extend a partial solution (beginning with the depot) by greedily taking the lowest-cost outgoing edge. This serves as a reasonable non-learned baseline for our approach. Motivated by the sampling-based decoding in our approach, we additionally define a sampling version of the NN baseline, called NN-softmax, which extends its solution by selecting the next edge with probability inversely proportional to edge cost. The NN-softmax baseline samples and evaluates many solutions, and returns the best among them.

Heuristic: GILS-RVND. GILS-RVND [9] is a metaheruristic that combines Greedy Randomized Adaptive Search Procedures (GRASP) [21], Iterated Local Search (ILS) [22], and Variable Neighborhood Descent with Random neighborhood ordering (RVND) [23]. The framework has two nested loops. In each iteration of the outer loop, an initial solution is selected using a constructive procedure, and is then passed to an iterative local search procedure in the inner loop. In each iteration of the inner loop, we use RVND to find a better neighboring solution, then perturb the solution until the best current solution cannot be improved after a fixed number of steps. We use five traditional TSP neighborhood structures: Swap, 2-opt, Reinsertion, Or-opt-2, Or-opt-3. We can control the diversity of solutions generated by the constructive procedure, which allows us to start the iterative local search from different starting points. At the end, the best solution ever found is returned. Our implementation extends the open-source GitHub repository in [24].

4.3 Evaluation metrics

We evaluate the performance of our proposed approach in terms of the quality of solutions it generates as well as overall runtime in a number of ways. First, in order to evaluate the quality of the solutions relative to the optimal, we measure optimality gap against solutions obtained using CPLEX on relatively small graphs, and contrast the runtimes of the different approaches. Second, since our proposed method is a construction heuristic and is able to scale to larger problem sizes, we compare its runtime and solution costs against the GILS-RVND and NN heuristics. For each graph size, we report the performance on 500 test instances. At test time, we use the greedy decoding strategy, where we choose the best action, or the sampling strategy, where we sample 1280 solutions and take the best one. More sampling can possibly improve the solution at the cost of runtime. We also evaluate the result of combining greedy decoding with iterative local search with the same performance and runtime metrics.

4.4 Computing setup

We run all our experiments on a Linux Desktop with an Intel i9-9980XE CPU @ 3.0GHz and one Nvidia Titan RTX GPU. We train the RL+GNN policy on one GPU with 1.28M training instances per epoch over 140 epochs (for all graph sizes except 100 nodes, which we train for 200 epochs) and measure the inference runtime on CPU for a fair comparison with the heuristics/cplex baseline. We use a validation set of size 10000 to prevent overfitting. Training takes less than 1 day for graphs with less than 30 nodes and approximately 50 hours for graphs with 100 nodes. We implement the nearest neighbour heuristics with Python and Numpy and we run for 1280 iterations, the same number as the sampling-based decoding in RL. The GILS-RVND heuristic is implemented in C++ and compiled with g++7. We run the construction heuristics (outer loop) for 10 steps and run the iterative local search (inner loop) until there is no improvement for n steps (n is the total number of nodes). The implementation of the iterative local search used to improve RL solution (section 3.3) is the same as one inner loop of GILS-RVND.

5 Experimental results

Table 1 shows the overall performance and runtime of our method compared to the baselines. We split the methods into two categories - 1) sampling-based methods (nearest neighbor with softmax, GILS, and sampling-based decoding RL, RL-ILS) or 2) greedy construction methods (nearest neighbor and greedy-decoding RL). Sampling-based methods are typically expected to perform better since multiple solutions are sampled and only the best is selected. We provide a more detailed analysis of the results below.

5.1 CPLEX MIP

We were able to solve the MIP formulation with CPLEX on graphs of between 10 and 25 nodes. See the grey column in Table 1 for detailed results. As expected, we find that the time taken to find the optimal solution scales very rapidly as a function of the graph size. For larger graphs, CPLEX took too long to generate good quality solutions. Although we could use the intermediate CPLEX solutions as a baseline, the performance was poor compared to heuristics like nearest neighbor or GILS.

5.2 Heuristics

In contrast to the optimal CPLEX solver, we are able to run both the nearest neighbor and GILS-RVND heuristics for all test graphs in a short period of time. The greedy NN heuristic runs extremely fast on all test instances. The sampling-based NN heuristics can take up to 0.3 seconds on larger graphs, but they outperform the naive NN heuristic across all types of instances. The state-of-the-art MLP heuristic GILS exhibits stronger performance on larger test instances. However, GILS needs more iterations of local search for graphs with more nodes. While this certainly improves its performance, it requires much longer runtimes as the size of the graph increases.

5.3 RL + GNN Results

The method described in Section 3 has been implemented for symmetric and asymmetric MLP instances (uniformly distributed graphs sampled in the unit square in 2d-euclidean space with different service times). An example of the training curves is shown in Figure 3, where the policy is trained on symmetric graphs with 20 nodes for 350k iterations (140 epochs with 2500 iterations per epoch). Early stopping is performed on all training runs to save the policy with the lowest average validation cost.

As shown in Table 1, the policy is tested on a holdout test set of 500 MLP graph instances of $n =$ 10, 20, 30, 50 and 100 using both the greedy and sampling-based decoding strategies described in Section 3. The optimal solutions for small graphs (with 10, 20 nodes) are obtained using CPLEX. The greedy construction outperforms nearest neighbour across all graph sizes and graph types in terms of the total cost. Its solutions are significantly closer to optimal results than nearest neighbour. When comparing sampling-based methods, we obtain better results than nearest neighbour with softmax and GILS-RVND on small graphs with 10 and 20 nodes. While the sampling-based decoding performs

Table 1: The average costs and runtimes from evaluating the trained RL+GNN policies and the baselines on 500 randomly generated MLP graph instances for each node size n=10,20,30,50,100 and each dataset class S0,S1,S2. RL-a indicates greedy decoding, RL-b indicates sampling-based decoding and RL-c indicates greedy RL + iterative local search. NN-b is nearest neighbour with softmax. Bolded entries indicate the best performance within a category (i.e. sampling vs greedy). Dashes indicate unavailable information.

	Size	Total Latency							Runtime (s)						
		NN-b	GILS	RL-b	RL-c	NN	RL-a	CPLEX	NN-b	GILS	RL-b	RL-c	NN	RL-a	CPLEX
S0	10	12.50	12.51	**12.37**	**12.37**	12.98	**12.39**	12.36	0.005	**0.001**	0.040	0.012	**0.001**	0.010	0.137
	20	34.16	34.07	**33.75**	33.77	36.40	**34.08**	33.63	0.012	**0.005**	0.121	0.019	**0.001**	0.016	3.225
	30	61.06	60.54	60.29	**60.24**	66.07	**61.44**	-	0.024	**0.019**	0.222	0.027	**0.001**	0.022	-
	50	129.3	**126.5**	128.5	**126.5**	141.23	**131.9**	-	0.067	0.126	0.620	**0.053**	**0.001**	0.035	-
	100	365.2	**346.1**	357.0	346.6	395.03	**368.6**	-	**0.256**	1.872	2.826	0.269	**0.001**	0.073	-
S1	10	22.88	22.60	**22.46**	22.47	23.66	**22.50**	22.44	0.005	**0.001**	0.041	0.013	**0.001**	0.010	0.140
	20	83.60	81.05	**80.76**	80.80	88.14	**81.25**	80.64	0.013	**0.005**	0.117	0.020	**0.001**	0.016	2.975
	30	179.6	**170.1**	170.8	170.7	190.9	**172.3**	-	0.024	**0.020**	0.238	0.028	**0.001**	0.023	-
	50	479.9	444.2	447.9	**444.1**	511.3	**452.7**	-	0.067	0.131	0.614	**0.053**	**0.001**	0.036	-
	100	1851	**1635**	1669	**1635**	1965	**1697**	-	**0.258**	1.966	2.826	0.276	**0.001**	0.073	-
S2	10	57.58	56.21	**56.02**	56.03	58.97	**56.08**	56.00	0.005	**0.001**	0.043	0.012	**0.001**	0.009	0.139
	20	242.4	232.7	232.4	**232.3**	251.2	**232.9**	232.2	0.013	**0.005**	0.118	0.020	**0.001**	0.016	2.071
	30	557.4	528.6	528.5	**528.2**	578.7	**530.2**	-	0.024	**0.020**	0.222	0.028	**0.001**	0.023	-
	50	1575	**1476**	1478	**1476**	1633	**1484**	-	0.067	0.129	0.605	**0.052**	**0.001**	0.035	-
	100	6492	**5956**	5985	**5956**	6724	**6010**	-	**0.256**	1.957	2.816	0.272	**0.001**	0.073	-

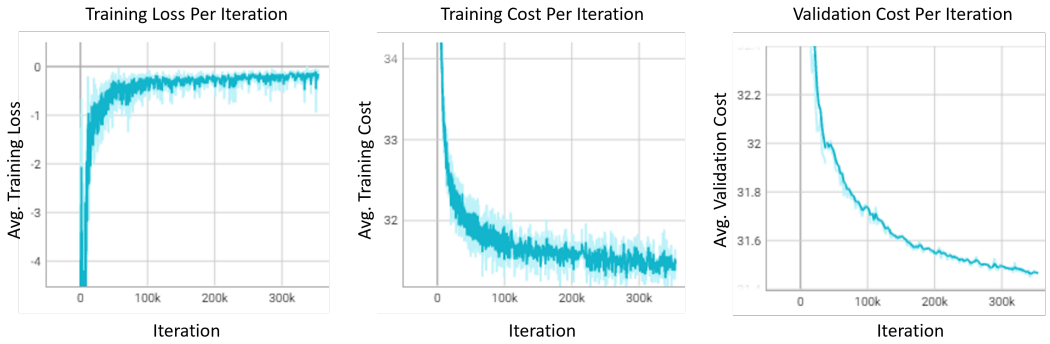

Figure 3: **Training loss and training/validation cost** over 140 epochs (350k iterations) for solving MLP on randomly generated (uniformly distributed) graphs with 20 nodes. (left): evolution of training loss; (middle): evolution of training path cost; (right): evolution of validation path cost.

slightly better in terms of average path cost, its average duration is higher than that of the greedy decoding. This trade-off can be tuned by changing the number of samples used in the probabilistic decoding. However, the key takeaway is that on small graphs, the trained policy produces solutions which are competitive with state-of-the-art domain-specific methods.

5.4 Greedy decoding RL + ILS

We also demonstrate that greedily-decoded RL solutions can be significantly improved with iterative local search. This method is referred as RL-c in Table 1. The performance (total latency) of RL+ILS is on-par with that of GILS-RVND and sampling-based decoding across all graph types and graph sizes. While it is slightly slower compared to only greedy decoding, the run time scales significantly better compared to GILS or sampling-based decoding. The key message is that combining RL with ILS can be very effective at reducing the time taken by sampling-based decoding while maintaining or even exceeding in terms of performance.

5.5 Optimality Gaps

On the left subplot of Figure 4, we plot the optimality gap as a function of the size of the graph on S0. Sampling-based RL achieved a very low optimally gap (< 1%) and outperformed all other heuristics for graphs with up to 25 nodes. On the right subplot, we compare the optimality gap against run time on symmetric instances of size 25. The bottom-left corner represents the best trade-off. At the two extremes, we have the nearest neighbor heuristic which is fast but less optimal, and the rl-sampling approach which is two orders of magnitude slower but also two orders of magnitude closer to optimal. While rl-greedy does not perform as well compared to rl-sample, improving its solutions with iterative local search (rl-ils) can significantly improve the performance with a small computational overhead.

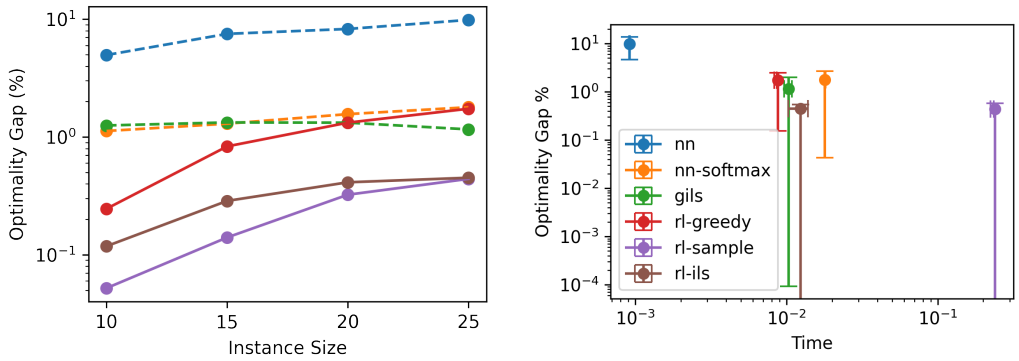

Figure 4: (Left) Average optimality gap as a function of instance size on symmetric instances (S0); (Right) Distribution of optimality gap vs time on symmetric instances (S0) of size 25. Markers indicate the median, and error bars indicate the range of the middle 50 percentile of instances. Note: Truncated error bar for rl-sample and rl-ils indicates the lower 25 percentile were solved to optimality (and thus have zero optimality gap).

5.6 Generalization to Graphs of Different Size

We also evaluate the ability of policies trained on a specific graph size to generalize to solving MLP problems over graphs of different size. In practice, the problems that might be encountered would likely have a range of sizes, depending on the use case. We would like to see that the models produce reasonable solutions, even on graph sizes which they haven't been trained on explicitly. The top subplot of Figure 5 shows the result of this experiment. We find that there is significant deterioration in the quality of solutions as the test size gets further away from training size. However, we also see that even in such cases, the performance is often still superior to that of the nearest neighbor heuristic.

5.7 Generalization to Graphs of Different Type

We evaluate the ability of the model trained on asymmetric graphs to generalize to symmetric graphs. We test the models trained on S1/S2 instances, where the service times are non-zero, on the S0 test instances. As shown in the bottom subplot of Figure 5, the model trained on S1 instances experiences less performance drop than the model trained on S2 instances when tested on symmetric graphs. This is because the instances in S0 could be seen as a special case of S1 instances where all the service times are zero, and the S2 instances have larger service times and are thus more "out-of-distribution". The model trained on S1 instances outperforms the nearest neighbor baseline for graphs with up to 50 nodes.

6 Conclusion

In this paper, we explore the application of reinforcement learning to derive a domain-specific construction heuristic for the Minimum Latency Problem. Following prior work, we parametrize the heuristic as a graph attention network [16], and optimize it by gradient descent using the REINFORCE [19] gradient estimate. We extend this method to support asymmetric graphs, which are

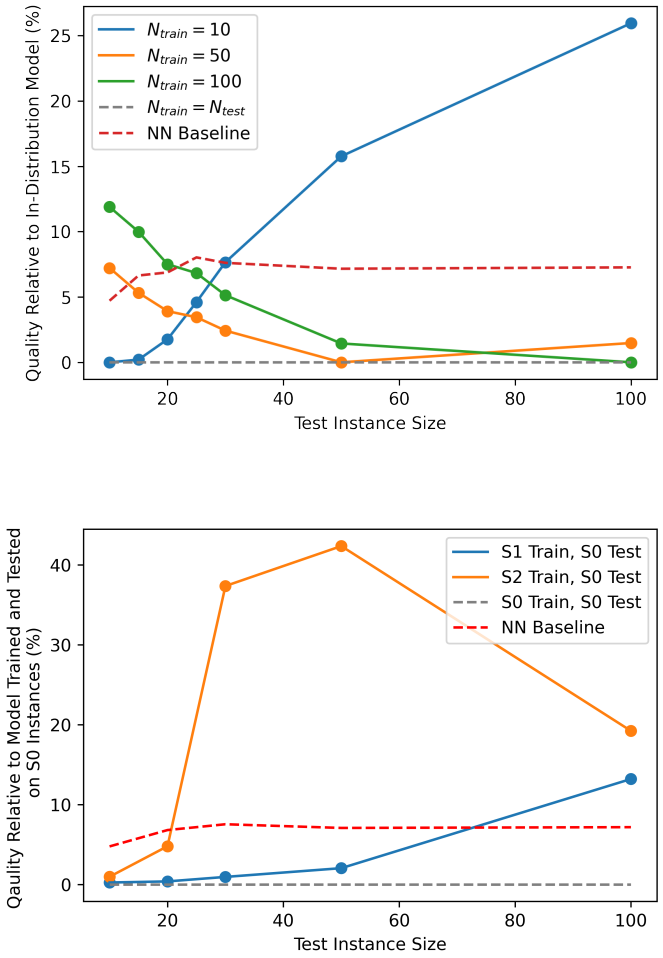

Figure 5: Solution quality on out-of-distribution data. On the x-axis, we vary the size of the graphs in the test-set. The y-axis represents the average gap in solution quality relative to that of the "in-distribution" model (i.e. the model with $N_{train} = N_{test}$ in the top subplot, and the model trained and tested on S0 instances in the bottom subplot). Only the greedy decoding is used, so we compare to the NN baseline shown as the red dotted line. (Top) Solution quality evaluated on different graph sizes. (Bottom) Solution quality evaluated on different graph types.

more representative of the travelling deliveryman/repairman instantiations of the MLP. We evaluate the approach extensively against a number of baselines, ranging from exact solvers to state-of-the-art domain-specific heuristics, and show that it produces consistently good solutions across a large test set with different graph types and sizes. We additionally propose to combine the approach with local search methods and show that this can yield even higher quality solutions for larger graph sizes, without adding unreasonable computational overhead. Furthermore, our tests demonstrate an ability of the learned policies to generalize to moderately out-of-distribution data, both in terms of graph size as well as structure. Our work shows that learning to solve graph optimization problems with RL + GNN can be a promising idea, particularly for those applications that have not been studied extensively.

References

[1] F. Angel-Bello, A. Alvarez, and I. García, "Two improved formulations for the minimum latency problem," *Applied Mathematical Modelling*, vol. 37, no. 4, pp. 2257–2266, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0307904X12003459

[2] F. Afrati, S. Cosmadakis, C. H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou, "The complexity of the travelling repairman problem," *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications*, vol. 20, no. 1, pp. 79–87, 1986.

[3] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, "The minimum latency problem," in *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '94. New York, NY, USA: Association for Computing Machinery, 1994, p. 163–171. [Online]. Available: https://doi.org/10.1145/195058.195125

[4] M. Fischetti, G. Laporte, and S. Martello, "The delivery man problem and cumulative matroids," *Oper. Res.*, vol. 41, pp. 1055–1064, 1993.

[5] van Ca Cleola Eijl, "A polyhedral approach to the delivery man problem," 1995.

[6] I. Méndez-Díaz, P. Zabala, and A. Lucena, "A new formulation for the traveling deliveryman problem," *Discrete applied mathematics*, vol. 156, no. 17, pp. 3223–3237, 2008.

[7] H. Abeledo, R. Fukasawa, A. Pessoa, and E. Uchoa, "The time dependent traveling salesman problem: polyhedra and algorithm," *Mathematical Programming Computation*, vol. 5, no. 1, pp. 27–55, 2013.

[8] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar, "Paths, trees, and minimum latency tours," in *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, January 2003, p. 36. [Online]. Available: https://www.microsoft.com/en-us/research/publication/paths-trees-and-minimum-latency-tours/

[9] M. M. Silva, A. Subramanian, T. Vidal, and L. S. Ochi, "A simple and effective metaheuristic for the minimum latency problem," *European Journal of Operational Research*, vol. 221, no. 3, pp. 513–520, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037722171200269X

[10] Ítalo Gomes Santana, "Improving a state-of-the-art heuristic for the minimum latency problem with data mining," 2019.

[11] W. Zhang and T. Dietterich, "Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling," 06 2001.

[12] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers Operations Research*, vol. 134, p. 105400, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054821001660

[13] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2017.

[14] Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković, "Combinatorial optimization and reasoning with graph neural networks," 2021.

[15] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/d9896106ca98d3d05b8cbdf4fd8b13a1-Paper.pdf

[16] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" 2019.

[17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[19] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3–4, p. 229–256, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992696

[20] G. Reinelt, "TSPLIB–a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[21] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization*, vol. 6, no. 2, pp. 109–133, 1995.

[22] F. W. Glover and G. A. Kochenberger, *Handbook of metaheuristics*. Springer Science & Business Media, 2006, vol. 57.

[23] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & operations research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[24] R. Mendes, "Minimum latency problem (mlp)," https://github.com/renatamendesc/MLP, 2021.