Looking Ahead to Avoid Being Late: Solving Hard-Constrained Traveling Salesman Problem

Jingxiao Chen
timemachine@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Ziqin Gong
gongzq0301@gmail.com
Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China

Lvda Chen
chenlvda@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Minghuan Liu
minghuanliu@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Jun Wang
jun.wang@cs.ucl.ac.uk
University College London
London, UK

Yong Yu
yyu@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Weinan Zhang
wnzhang@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

ABSTRACT

Many real-world problems can be formulated as a constrained Traveling Salesman Problem (TSP). However, the constraints are always complex and numerous, making the TSPs challenging to solve. When the number of complicated constraints grows, it is time-consuming for traditional heuristic algorithms to avoid illegitimate outcomes. Learning-based methods provide an alternative to solve TSPs in a soft manner, which also supports GPU acceleration to generate solutions quickly. Nevertheless, the soft manner inevitably results in difficulty solving hard-constrained problems with learning algorithms, and the conflicts between legality and optimality may substantially affect the optimality of the solution. To overcome this problem and to have an effective solution against hard constraints, we proposed a novel learning-based method, MUSLA, that uses multi-step looking-ahead information as the feature to improve the legality of TSP with Time Windows (TSPTW) solutions. Besides, we constructed TSPTW datasets with hard constraints in order to accurately evaluate and benchmark the statistical performance of various approaches, which can serve the community for future research. With comprehensive experiments on diverse datasets, MUSLA outperforms existing baselines and shows generalizability potential.

CCS CONCEPTS

• **Computing methodologies** → *Machine learning*; **Artificial intelligence**.

DAI '24, Dec 18th – Dec 22th, 2024, Singapore

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

KEYWORDS

combinatorial optimization, traveling salesman problem with time windows, deep learning

ACM Reference Format:
Jingxiao Chen, Ziqin Gong, Lvda Chen, Minghuan Liu, Jun Wang, Yong Yu, and Weinan Zhang. 2024. Looking Ahead to Avoid Being Late: Solving Hard-Constrained Traveling Salesman Problem. In *Proceedings of The Sixth International Conference on Distributed Artificial Intelligence (DAI '24).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

NP-hard combinatorial optimization problems play a vital role in modern practical applications and industries. In the real world, problems are always attached to a set of complex constraints, making them challenging to solve. Typically, the constraints in combinatorial optimization problems include hard and soft ones. Soft constraints tolerate slightly violating the constraints in a small range, whereas hard constraints strictly prohibit any violation.

In this paper, we focused on the set of popular Traveling Salesman Problems (TSPs), which are described as asking a salesman to visit each city with minimizing the total length of the tour. This scenario is commonly used in profit optimization in industrial production procedures. Considering different kinds of constraints, there are many variants of TSPs. For instance, Traveling Salesman Problems with time windows (TSPTW) is a famous hard-constrained variant of TSP in the vehicle routing problem (VRP) family, which constrains the salesman from visiting each city in a particular range of time and meanwhile minimizes the total length of the tour; In comparison, the capacitated vehicle routing problem (CVRP) puts soft constraints on TSP which require multiple salesmen, also called vehicles, with limited carrying capacity to deliver items to various locations.

Traditional heuristic searching approaches, such as LKH3 [8], traverse a large number of solutions and search for the best one that satisfies the constraints. The Searching method guarantees

finding feasible solutions under arbitrary constraints. However, searching-based algorithms require a well-designed heuristic function for a specific problem and consume a lot of time to search for different solutions for a new problem instance. In order to improve efficiency, recent work has turned to machine learning techniques that construct end-to-end solvers to generate high-quality solutions in a few trials and allow GPU acceleration to improve efficiency further. However, limited trials make it difficult for these solvers to find a feasible solution. Therefore, solving hard constraints with learning-based approaches becomes more challenging.

In order to deal with hard constraints, most learning-based methods train an end-to-end solver in the RL paradigm and relax the time windows as soft constraints [22] or solve a soft-constrained variant of TSPTW, *e.g.*, Traveling Salesman Problem with Time Windows and Rejections [27]. In contrast to RL methods requiring elaborate reward designs and millions of environment interactions, supervised learning (SL) can be easily trained with fixed offline expert datasets, which is more practical for real-world problems. However, SL was only used to solve regular TSPs in earlier research [9, 18, 25, 26], but it is rarely used for constrained ones. Challenges, such as obtaining information about constraint boundaries from expert data, prevent applying the SL method to hard-constrained TSPs.

In this paper, we proposed a novel algorithm named MUlti-Step Look-Ahead (MUSLA) to efficiently resolve one of the hard-constrained TSPs, TSPTW. In detail, MUSLA introduced a novel one-step look-ahead way to gather future information regarding constraint boundaries and train a policy π^{+1} by imitating an expert. Based on the well-trained policy, we augment expert datasets with multi-step look-ahead information collection. The gathered information about future situations provides a better perception of the constraint boundaries. With the enhanced dataset, we further train the MUSLA policy π^{+m}.

In a nutshell, our main technical contributions are threefold:

- We propose MUSLA, which includes a novel looking-ahead mechanism based on supervised learning methods. MUSLA enhanced optimality and legality of SL solutions by augmenting datasets with searched information.
- We design two kinds of TSPTW datasets to evaluate the performance of solvers better.
- Compared to state-of-the-art work with RL paradigm, MUSLA outperforms other baselines and has a good balance between solution quality and validation rates.

2 RELATED WORK

Recent learning-based work solved TSP and its variants in a reinforcement learning paradigm. Graph Neural Networks [4, 13, 24] and Attention mechanism [23] are the major architectures in state-of-the-art work. Kim *et al.* [12], Kwon *et al.* [15] leverage symmetricities to improve the generalization capability of learning-based solver. Recent work [24] has achieved comparable performance to traditional methods on a scale of no more than 200 nodes. On large-scaled TSPs, learning-based methods even demonstrated faster execution with slightly dropped performance [7]. In the RL architecture, the learning-based policy trials and errors in sampled problem instances and updates the policy according to reward signals. The reward function is usually set to be the negation of the tour length.

For constrained TSP, such as Traveling Salesman Problem with Time Windows, solving constraints is treated as another optimization objective, which brings challenges to learning-based methods. Ma *et al.* [17] proposed a hierarchical RL policy to separate constraints from original optimization objectives. Alharbi *et al.* [1] used a complex hybrid network architecture to improve solution quality. Another feasible option is to solve reductions of TSPTW. Tang *et al.* [22] relaxed time windows as soft constraints, and Zhang *et al.* [27] solved a soft-constrained variant of TSPTW. Other methods overcome the challenge of constraints by combining learning-based methods with traditional algorithms [5, 19, 28], but at the cost of the computational burden. For RL approaches, there are usually two drawbacks to solving constrained TSP. One is the additional objective requires well-designed reward shaping to balance legality and optimality. The other is that RL methods require a lot of trial-and-error and interactions with the environment during training, resulting in high training costs especially for real-world problems.

Supervised learning (SL) is an alternative learning-based method to solve TSP [9, 18, 25, 26]. The SL policy is trained on datasets labeled by experts, thus avoiding additional trial and error and reward design. Recent work usually uses exact algorithm [2, 21] and heuristic algorithms [8] as oracles to generate the datasets.

3 PROBLEM FORMULATION

3.1 Traveling Salesman Problem with Time Windows

We first introduce the traveling salesman problem with time windows (TSPTW) and describe the challenge in a hard-constrained setting. Consider a symmetric complete graph $\mathcal{G} = (V, E)$, where the set of nodes $V = \{0, 1, \ldots, n\}$ denotes the set of *cities*, and the set of edges $E \subset V \times V$ denotes the set of *paths* between every two *cities*. Each *city* $i \in V$ has two attributes, the 2-D coordination a_i and the visiting time window $[t_i^s, t_i^e]$. The length of edge $e_{i,j} \in E$ is $L_{i,j} = ||a_i - a_j||_2$, where $||\cdot||_2$ is the l_2 norm. The goal of TSPTW is to minimize the distance of the total path by asking the *salesman* to visit all the *cities* exactly once and return to the start *city* under the constraint that each *city* must be visited within the given time window. A formal definition is as follows:

$$\begin{aligned} &\min_{X=\{x_0,x_1,x_2,\ldots,x_n\}} L_{x_n,x_0} + \sum_{i=0}^{n-1} L_{x_i,x_{i+1}} \\ &s.t.\ t_{x_i}^s \le t_i \le t_{x_i}^e, \\ &\quad t_i = \max\{t_{i-1} + L_{x_{i-1},x_i}, t_{x_i}^s\} \end{aligned} \tag{1}$$

where X is a solution tour, and t_i is the time to visit city x_i. For simplicity, we presume the speed number is equal to 1 and exclude the speed element from the calculation. A legal X is a permutation of nodes, indicating $x_i \neq x_{i'}, \forall i \neq i'$. Without loss of generality, we assume $x_0 = 0$ and traveling time between x_i, x_{i+1} is $L_{x_i,x_{i+1}}$.

A hard constraint is one that must be satisfied at all times. It is notable that $t_i \le t_{x_i}^e$ is a hard constraint, while $t_i \ge t_{x_i}^s$ is not. If city x_i is visited before the earliest time constraint $t_{x_i}^s$, the salesman should wait until $t_{x_i}^s$, *i.e.*, $t_i = \max\{t_{i-1} + L_{x_{i-1},x_i}, t_{x_i}^s\}$.

The goal of the TSPTW could also be minimizing the total tour duration, also called make-span, rather than the distance of the tour [11]. However, optimizing toward the make-span optimality may also reduce violations of constraints. It is difficult to design a reasonable evaluation dataset with the goal of make-span to clarify the model's ability to balance optimality and legality, which makes us ignore this situation.

As a hard-constrained problem, TSPTW is commonly regarded as a multi-objective problem in current learning-based paradigms. The optimal objective minimizes the tour distance and the legal objective minimizes the violation of constraints. In contrast to the soft-constrained problem where legality is considered a secondary optimization objective, optimality and legality should share the same status in a hard-constrained setting. Different from other constrained combinatorial optimization problems, such as CVRP and Vehicle Routing Problem with Time Windows (VRPTW), the legality of the solution in TSPTW can not be satisfied by feasible masking.

3.2 Supervised Learning with Route Construction

In recent learning-based work, solutions of TSP and its variants are modeled as an n-step route construction process [14]. Similar to the process of salesman travel, the solution sequence X is generated step by step in order with a learning-based policy π_θ. Given the current partial tour $X_{0:i} = \{x_0, \ldots, x_i\}$ and property of problem instance g, policy π_θ captures $p(x'|X_{0:i}, g)$, *i.e.*, the probability of visiting the next node x_i at step i. After n steps of route construction, the solution tour $X = \{x_0, x_1, \ldots, x_n\}$ is given by $p(X|g) = \prod_{i=0}^{n-1} p(x_{i+1}|X_{0:i}, g)$.

In order to train the policy π_θ, an expert dataset $\mathcal{D}$ is required, which includes multiple pairs of TSPTW instances g and expert solutions X^*. For each problem instance $g \sim \mathcal{D}$, the expert solution X^* is generated by a high-quality expert solver $X^* = \pi_{\text{expert}}(g)$. Hence the policy is trained to imitate the expert policy, whose objective, written formally, is maximum likelihood estimation:

$$\min_\theta \text{Loss}(\theta) = -\log p_{\pi_\theta}(X = X^*|g). \tag{2}$$

4 METHODOLOGY

In this section, we introduce our looking-ahead method for TSPTW. The pipeline of training is visualized in Figure 1. At first, we redesign dynamic features for TSPTW and add a dynamic encoder module for our SL model. Then, we augment the expert dataset with the one-step look-ahead mechanism and train a supervised learning policy π_θ^{+1}. Utilizing the policy, we gathered multi-step look-ahead (MUSLA) information to refine the expert datasets further and train the MUSLA policy π_θ^{+m}. Finally, we introduce a technique that can better adapt MUSLA policy to specific problem instances at inference time by modifying dynamic information.

Our method employs supervised learning instead of reinforcement learning due to two primary factors. At first, RL requires additional reward shaping to balance the optimality and legality of solutions. Secondly, applying MUSLA to RL necessitates the repeated collection of augmented information throughout the entire learning process, resulting in an unacceptable computing burden.

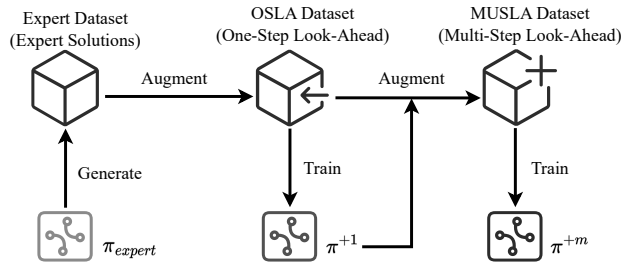


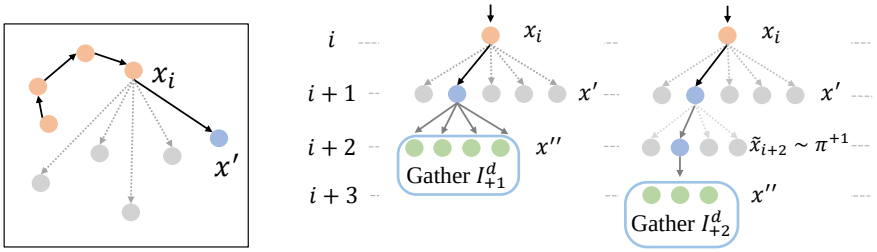
Figure 1: Method pipeline of MUSLA. We labeled expert datasets with LKH3 solutions in order to train a faster learning-based solver. With one-step look-ahead augmented datasets, we trained OSLA policy π^{+1}. OSLA policy directed further multi-step look-ahead data augmentation, resulting in MUSLA policy π^{+m}.

4.1 Learning with Dynamic Information

We consider a supervised learned policy π_θ using existing expert datasets, which consists of an encoder and a decoder. The encoder receives path-building information and generates embedding, and the decoder takes embedding as input and generates subsequent nodes. Similar to Alharbi *et al.* [1], π_θ takes static information $I^s(g) = \{a_i, t_i^s, t_i^e\}$ and history embedding $I^h(X_{0:i})$ of the current tour $X_{0:i}$ as input. However, this does not explicitly describe each possible next node x' on current solution $X_{0:i}$. It may cause cumulative errors in the time and distance for the next step selection, resulting in incorrect estimation for time window constraints. To alleviate this challenge, we design an additional dynamic node feature $I^d(X_{0:i}, x', g)$. In particular, the dynamic feature of each unvisited node $x' \in V \setminus X_{0:i}$ includes differences between x_i and x' in location and time dimensions. For example, in location dimension, features $\{L_{x',x_i}, a_{x'} - a_{x_i}\}$ describe the distance and direction between x' and current node x_i. In time dimension, features $\{t_{x'}^s - t_i, t_{x'}^e - t_i\}$ describe the time difference between time windows of x' and x_i. With dynamic information, visiting probability of the next node x' can be written as $p(x = x'|X_{0:i}, g) = \pi_\theta(x', I^s(g), I^h(X_{0:i}), I^d(X_{0:i}, x', g))$. In the subsequent representations, we streamline I^s and I^h, resulting in the simplification of the policy as $\pi_\theta(x', I^d)$. Details of feature design and model architecture are further described in supplementary materials.

4.2 Dataset Augmentation

Different from soft-constrained settings, violations of hard constraints result in complete failure. Hence, the major challenge of solving a hard-constrained TSPTW with the supervised learning policy π_θ is to learn implicit information about constraint boundaries. Since only the legal solutions are preserved in expert datasets, the learned policy may lack sufficient information to determine when the time constraints are violated. To provide constraint information and train a robust policy, our solution is to augment datasets using a look-ahead mechanism. Depending on the expanded step, we introduce the one-step look-ahead mechanism, based on which we further propose the multi-step look-ahead mechanism.



(a) Rout Construction (b) One-Step Look-Ahead (c) Multi-Step Look-Ahead

Figure 2: Illustration of the multi-step look-ahead mechanism for $m = 1$. Subfigure (a) shows the route construction at step i. Subfigures (b) and (c) illustrate the process of information gathering. Orange nodes have been determined to be in the current route $X_{0:i}$. Blue nodes are temporarily added to the route during the search. Future information is gathered from green points.

Figure 2 presents a schematic diagram of our approach. To ascertain if node x' should be the next node, we can enumerate all feasible routes that include x' by brute force. However, this approach is obviously impractical given the vast number of potential solutions. Hence, based on node x', we try m subsequent steps to construct an illusory partial solution with a length of $i + 1 + m$. By exploring several illusory partial solutions of x', we gather m-step look-ahead information I^d_{+m} as the policy's input.

4.2.1 One-Step Look-Ahead. We start from the one-step look-ahead (OSLA) mechanism. OSLA expands information on each node by trying to construct solutions one step ahead. At step i, we iterate each unvisited node x' to construct an illusory $i+1$ partial solution $X' = \{x_0, \ldots, x_i, x'\}$ and gather information for all unvisited node $x'' \in V \setminus X'$ based on the imaginary $i + 1$ step. The gathered information $I^d_{+1}(X', g)$ is used as additional dynamic features of node x' and helps determine the probability of choosing x' at the current step.

More specifically, the OSLA information includes two types of value. In the first type of feature, we select the set of nodes that are already to be late, i.e. $X''_{\text{late}} = \{x'' \in V \setminus X' | t_{i+1} + L_{x',x''} > t^e_{x''}\}$. We used the number of late nodes $|X''_{\text{late}}|$ and the maximum late time as features to capture the constraint violations of selecting x'. In the second type of feature, assuming $X''_{\text{late}} = \emptyset$, we add distance and time overhead when greedily visiting node x'' with minimum time overhead.

With the OSLA information as additional input, we can train an OSLA policy $\pi^{+1}_\theta(x', I^d, I^d_{+1})$ by supervised learning which is able to learn constraint boundaries by possible timeouts in the future. It is notable that future information is gathered based on expert solutions and does not impose an additional burden on the training process.

4.2.2 Multi-Step Look-Ahead with An OSLA Policy. Searching with one step helps to capture the constraint boundaries, but the collected information is still limited. The effect of a choice may require successive attempts to judge. To gather more guidance information from the future at an acceptable computational cost, we propose a multi-step look-ahead (MUSLA) mechanism to augment expert datasets further.

As the number of steps increases, potential solutions increase geometrically. In order to ensure the feasibility of calculating, we leveraged a pre-trained OSLA policy to screen for meaningful choices. Different from one-step augmentation, we only gather future information for the top k nodes x' with the highest probability of expert selection. Here we use policy π^{+1}_θ as an approximation to the expert strategy. For a specific node x', we continually construct m steps to get an illusory $i + 1 + m$ partial solution $\tilde{X}' = \{x_0, \ldots, x_i, x', \tilde{x}_{i+2}, \ldots, \tilde{x}_{i+1+m}\}$. After augmentation, we also gather information $I^d_{+2}(\tilde{X}', g)$ for all unvisited node $\tilde{x}'' \in V \setminus \tilde{X}'$ as the MUSLA features of x'.

This paper used a hyper-parameter setting of $k = 5, m = 1$ to gather multi-step features. Augmenting expert datasets with dynamic features I^d, I^d_{+1} and I^d_{+2}, we train the MUSLA policy, formulated as $\pi^{+2}_\theta(x', I^d, I^d_{+1} I^d_{+2})$.

4.3 Trade-off between Optimality and Legality

After training on augmented expert datasets, we obtain a robust MUSLA policy that balances optimality and legality. However, by modifying dynamic features, we can still adjust the balance of the two goals and adapt to specific TSPTW instances at the inference stage. For example, for a problem instance with tight time windows, the model might give an illegal tour that times out slightly. By modifying the tour time t_i with $t'_i = t_i + \epsilon$, the dynamic feature of the remaining time to reach a node $t^e_{x'} - t'_i$ becomes smaller, so the model adopts a more conservative strategy to avoid timeouts. Following this idea, we extend MUSLA to MUSLA-adapt. MUSLA-adapt tries different time offset values $\epsilon \in \mathcal{E}$ during inference and chooses the optimal legal solution as the final solution. Specifically, our experiments use $\mathcal{E} = \{-2, -1, -0.5, 0, 0.5, 1\}$. Algorithm 1 describes the route construction process of MUSLA-adapt.

5 IMPLEMENTATION: NETWORK STRUCTURE AND DATASETS

In this section, we introduce the implementation details of MUSLA, including network structure and datasets. For the network structure, we encode the gathered information using a transformer and graph network. We also create two kinds of TSPTW datasets, Medium and Hard, to demonstrate the effectiveness of our method.

Algorithm 1: Multi-Step Look-Ahead Adapt

Function MUSLA-adapt($\pi_\theta^{+1}, \pi_{\theta'}^{+2}, \mathcal{E}$):
  for $\epsilon \in \mathcal{E}$ **do**
    Initialize $X \leftarrow \{0\}$;
    for $i = 0$ **to** $n-1$ **do**
      Construct dynamic featrue I^d based on X, ϵ;
      $I_{+1}^d \leftarrow$ OslaGather(X, ϵ) ;
      Calculate probability $p(x') \leftarrow \pi_\theta^{+1}(x', I^d, I_{+1}^d)$;
      $I_{+2}^d \leftarrow$ MuslaGather(X, p, ϵ) ;
      Calculate $p(x') \leftarrow \pi_{\theta'}^{+2}(x', I^d, I_{+1}^d, I_{+2}^d)$;
      $X \leftarrow X \oplus \{x'\}, x' = \arg\max_{x'} p(x')$;
    end
    Update best solutions
     $X_{\text{best}} \leftarrow$ SelectBetter(X, X_{best}) ;
  end
  return *best solutions* X_{best};
Function OslaGather(X, ϵ):
  Initialize I_{+1}^d ;
  foreach *node* $x' \in V \setminus X$ **do**
    $X' \leftarrow X \oplus \{x'\}$;
    Update I_{+1}^d based on X' and ϵ ;
  end
  return *OSLA feature* I_{+1}^d ;
Function MuslaGather(X, p, ϵ):
  Initialize I_{+2}^d ;
  $V_s \leftarrow \{x'$ with first k-th highest $p(x')\}$;
  foreach *node* $x' \in V_s$ **do**
    $X' \leftarrow X \oplus \{x'\}$;
    Update I_{+2}^d based on OslaGather(X', ϵ) ;
  end
  return *MUSLA feature* I_{+2}^d ;

5.1 Network Structure

Following the recent learning-based TSP paper, such as Kool *et al.* [14], the end-to-end policy π_θ consists of an encoder and decoder, parameterized by θ. The encoder takes information needed to build the path as input and produces embedding. Then decoder takes embedding as input and produces the next node x_i. Different from recent works[1] which take static information $\{a_i, ts_i, te_i\}$ and history embedding of $X_{0:i}$ as input of policy, we additionally design dynamic node features to describe current solution $X_{0:i}$. Figure 3 presents the network structure of the policy.

In order to encode static, dynamic, and historical information, we construct the encoder with three model structures. The first part, the static graph encoder $e_s(\cdot)$, encodes the graph structure of the current TSPTW instance with graph attention networks [10]. The graph encoder aggregates neighbor information across nodes and captures the graph structure of nodes for a specific TSPTW instance g. However, the powerful graph neural network structure also brings a huge computational overhead. In order to ensure that encoding dynamic information at each step does not bring too much computational burden, we apply the second part with a node-wise MLP $e_d(\cdot)$ for the dynamic node feature. Given the embedding of encoded static and dynamic features at step i as h_x^s, h_x^i, the historical sequence of $X_{0:i}$ can be described as $H_{0:i} = \{(h_{x_0}^s,), (h_{x_1}^s, h_{x_1}^0), \ldots, (h_{x_i}^s, h_{x_i}^{i-1})\}$. As the third part of the encoder, we use a Gated Transformer-XL [20] $e_h(\cdot)$ to model the historical embedding sequence. The encoder can be formulated as follows,

$$h^s = e_s(g),\ h_{x'}^i = e_d(x', h^s, X_{0:i}),\ h_i^h = e_h(H_{0:i}). \tag{3}$$

At each construct step i, our model predicts the probability of visiting each unvisited node with attention mechanism $d(\cdot)$ and soft-max. The queries come from historical embedding $H_{0:i}$ and the keys and values come from the dynamic embedding $h_{x'}^i$. The formulation for the probability of node x' at step i can be written as

$$p(x = x' | X_{0:i}, g) = \frac{d(h_{x'}^i, h_i^h)}{\sum_{j \in V \setminus X_{0:i}} d(h_j^i, h_i^h)},\ x' \in V \setminus X_{0:i}. \tag{4}$$

5.2 Dataset with Hard-Constraints

In order to highlight the ability of the algorithm to balance optimality and legality, we create two kinds of TSPTW datasets, Medium and Hard. Medium is designed with a clear random distribution that increases the difficulty of satisfying the time window constraints. Hard is constructed in a complex way and aims to evaluate the generalization ability of the model. The expert solutions of our datasets are given by LKH3 solver [8] and we only discuss the generation of problem instances $g \sim D$ in this section.

Traditional approaches, like LKH3, provide datasets with a limited number of problem instances, which is insufficient for training learning-based algorithms. Most recent learning-based TSPTW work did not follow a uniform way to generate data and ignored the importance of dataset quality. Cappart *et al.* [5] and Ma *et al.* [17] generated time windows following the visiting time t_i' of a given solution X'. Their datasets give too strong prior assumptions about the time window and also oversimplify this problem. A trivial greedy policy that takes unvisited node i with the smallest value $t_{x_i}^s$ is able to obtain a near-optimal solution with low violation of constraints. Zhang *et al.* [27] generated datasets with pure randomization, but inappropriate parameter settings lead to the absence of legal solutions for most problem instances.

We demonstrate detailed analysis in Appendix A. Based on the weaknesses of previous works, we propose datasets with proper constraints to highlight the challenge of balancing two objectives.

Medium dataset. Similar to Zhang *et al.* [27], we generate the random dataset Medium by randomly sampling coordinates a_i , time windows $[t_i^s, t_i^e]$ of $n+1$ nodes. The 2D-coordinates a_i are sampled uniformly in a grid of $\mathcal{U}[0, 100]^2$. The time windows are given by sampling start time t_i^s and width $t_i^e - t_i^s$ uniformly:

$$t_i^s \sim \mathcal{U}[0, T_n],\ t_i^e = t_i^s + T_n \cdot \mathcal{U}[\alpha, \beta] \tag{5}$$

where α, β are hyper-parameters, and T_n is the expected distance of an arbitrary TSP tour on $n+1$ nodes. For $n = 20$, a rough estimate of T_n is $T_{20} \approx 10.9$. By expanding the sampling range of the start time, the size of the feasible solution set in the dataset can be effectively limited, thus bringing conflicts between optimality and legality.

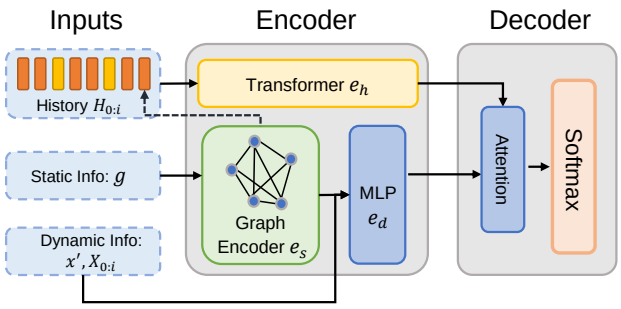


Figure 3: Network structure of our policy.

Hard dataset. In order to guarantee the credibility of the evaluation, we add a more difficult dataset HARD. In this dataset, we sample problem instances for training and evaluation from slightly different distributions. This setting is similar to the real-world situation, and there is often a certain deviation between the training and test scenarios. We test the model's generalization ability to different problem instances on this dataset.

In supplementary materials, we provide a detailed description and pseudocode of these constructed datasets.

Supplementary training dataset. Due to the incapacity to iterate through different problem instances in TSPTW, a learning-based algorithm trained on fixed datasets tends to overfit or be unstable when solving new problems. To alleviate the issue, we augment variants of the aforementioned datasets to stabilize the model performance. The supplementary dataset includes simplified datasets, such as removing one or two sides of time windows, and also complex datasets. For the purpose of elucidating our conclusion, we trained our model on all dataset variants but only presented evaluation results for Easy and Hard datasets. Details regarding additional datasets are provided in supplementary materials.

6 EXPERIMENTS

In this section, we show empirical results of evaluating our method on MEDIUM and HARD datasets with problem sizes of $N = 20, 50, 100$. We begin by evaluating the two objectives, legality and optimality, of different models, and contrasting hard and soft constraints. Then, we demonstrate the effect of various components of our methodologies.

6.1 Setups

Evaluation metrics. In order to compare method performance from different perspectives, we use four evaluation metrics:

- **Illegal rate** is the proportion of illegal solutions produced by the algorithm in datasets, which reflects the legality of the solutions.
- **Solution gap**, *i.e.*, optimality of solutions, is the gap between solution distance L and distance of expert solution, L_{expert}, calculated by $(L/L_{\text{expert}} - 1)$. In particular, we only calculate the solution gap for legal solutions.
- **Solving time**, the execution time each algorithm takes to solve 1 000 problem instances.
- **Total timeout**, the sum of timeouts on each node.

All the metrics are evaluated on test sets consisting of 1 000 instances.

Configuration. For fair comparisons, we evaluate solving time on the same hardware configuration. Greedy policies and heuristic policies execute on one Intel(R) Xeon(R) Platinum 8255C CPU @ 2.50GHz (with 8 cores). Learning-based policies execute on one NVIDIA GeForce RTX 3090. We trained the learning-based models for 500 000 samples and 100 epochs. Hyperparameters for training are listed in supplementary materials.

Baselines. We compare three types of baseline algorithms described as follows.

- **Heuristic baseline.** We consider the state-of-art TSPTW solver, *LKH3* [8], as the oracle to calculate the solution gap and generate datasets. Since we screened out a few instances that LKH3 cannot solve, the illegal rate of LKH3 is 0%.
- **Greedy baselines.** We provide two trivial rule-based greedy policies as poor baselines. Their results may be regarded as fair lower bounds of solver performance. Greedy policies follow the route construction process to generate the solution. At each step, *Minimum time-consuming greedy* (Greedy-MT) chooses the node with the minimum arrival time to visit. The calculation of arrival time includes the waiting time for the earliest access time t_i^s. *Minimum latest access time greedy* (Greedy-LT) chooses the node with the minimum latest access time t_i^e.
- **Learning-based baselines.** As mentioned in Section 5, most of the recent learning-based TSPTW approaches are evaluated on inconclusive datasets and did not release available code. We selected two RL works solving similar problems for our comparison. Both works provide open-source code. *JAMPR* [6] is the state-of-art for Vehicle Routing Problem with Time Windows (VRPTW), a variant of TSPTW with multiple salesmen. We adapt JAMPR for the TSPTW by removing additional constraints and allowing only one salesman. JAMPR resolves the VRPTW by successively constructing multiple routes and masking all timeout nodes for the current route. In TSPTW, the only route must visit every node without exceeding the timeout limit, and there is no simple, feasible masking that can be used to avoid the timeout. The other method is *AM* [14], which is a well-known route construction method. We adapt AM for TSPTW with additional time-window $[t_i^s, t_i^e]$ features as input. The TSPTW solutions adhere to a fixed order, thereby rendering algorithms that rely on solution symmetry, such as POMO [15], inapplicable to the TSPTW problem. The total reward function R for RL algorithms consists of route length, timeout period, and the number of timeout nodes,

$$R = R_{\text{route\_length}} + R_{\text{total\_timeout}} + R_{\text{number\_of\_timeout\_nodes}}.$$

6.2 Results and Analysis

Table 1 shows the experimental results on MEDIUM and HARD datasets. Overall, on different types and sizes of datasets, MUSLA and MUSLA-adapt outperforms other learning-based baselines by a large margin on the solution gap. However, many of the baselines show extreme imbalances in experiments. While all generated routes are legal, imbalanced policies show a large gap in optimality compared to the expert baseline. In order to better evaluate the

Table 1: The result table compares the performance of our model with other baselines.

Methods		Medium			Hard			Time(s)
		Illegal(%)	Gap(%)	Timeout	Illegal(%)	Gap(%)	Timeout	
$N = 20$	LKH3	0.00	0.00	0.00	0.00	0.00	0.00	0.2
	Greedy-MT	0.00	95.97	0.00	12.50	51.70	7.48	0.27
	Greedy-LT	0.00	128.82	0.00	5.13	168.57	4.70	0.08
	AM	5.34	16.22	0.03	83.00	52.13	5.15	0.28
	JAMPR	0.00	116.02	0.00	8.29	74.31	15.56	6.03
	MUSLA π^{+2}	3.73	**5.33**	0.26	5.20	**12.10**	0.26	3.30
	MUSLA adapt	0.20	**4.02**	0.24	0.40	**10.25**	0.10	19.72
$N = 50$	LKH3	0.00	0.00	0.00	0.00	0.00	0.00	11.64
	Greedy-MT	0.00	196.12	0.00	18.64	69.08	25.33	0.27
	Greedy-LT	0.00	257.49	0.00	17.19	311.07	89.21	0.08
	AM	9.90	32.68	0.09	49.50	65.88	9.08	0.27
	JAMPR	0.00	249.03	0.00	1.31	207.10	0.88	7.30
	MUSLA π^{+2}	8.20	**7.32**	1.80	18.90	**16.71**	4.42	7.63
	MUSLA adapt	0.10	**5.63**	0.99	3.10	**15.24**	2.22	45.97
$N = 100$	LKH3	0.00	0.00	0.00	0.00	0.00	0.00	7588.75
	Greedy-MT	0.00	314.04	0.00	20.42	79.25	36.03	0.30
	Greedy-LT	0.00	409.62	0.00	30.02	468.76	408.18	0.08
	AM	9.00	239.57	0.05	33.40	132.18	3.42	3.01
	JAMPR	100.00	N/A	14.82	100.00	N/A	734.44	9.53
	MUSLA π^{+2}	18.60	**14.60**	24.81	50.50	**37.05**	96.39	58.83
	MUSLA adapt	0.60	**12.01**	9.59	31.90	**35.59**	89.57	403.53

Table 2: Comparison for mentioned variants of our methods.

Methods		Medium			Hard			Time(s)
		Illegal(%)	Gap(%)	Timeout	Illegal(%)	Gap(%)	Timeout	
$N = 50$	Static	82.70	7.60	8.56	74.30	122.07	9.87	1.16
	Dynamic	50.30	6.16	0.64	55.60	31.41	4.12	1.36
	OSLA π^{+1}	11.80	8.15	3.53	24.50	18.55	8.43	1.56
	MUSLA π^{+2}	8.20	7.32	1.80	18.90	16.71	4.42	7.63
	MUSLA adapt	0.10	5.63	0.99	3.10	15.24	2.22	45.97
	OSLA-Medium	26.80	12.22	6.12	39.10	43.46	6.84	1.56

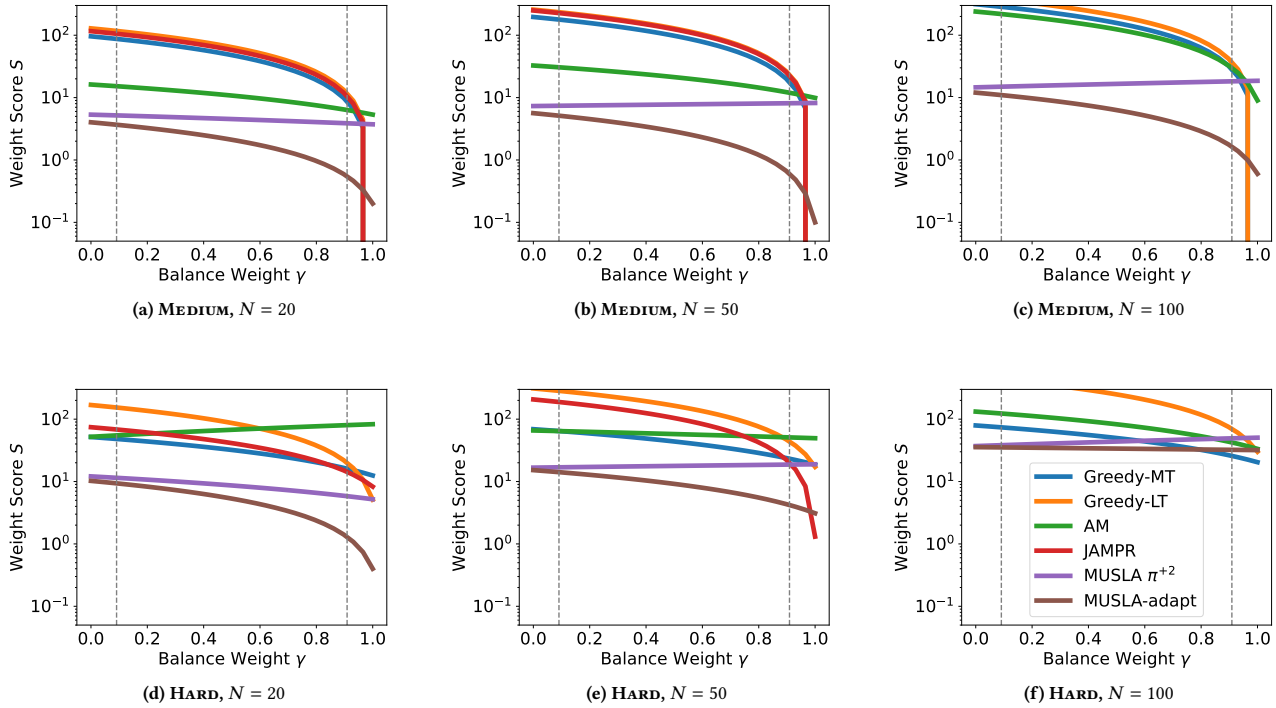
ability of the algorithm to balance the two indicators, we use the weighted score to quantify the performance. The weighted score S is calculated by

$$S = \gamma \cdot \text{Illegal}(\%) + (1 - \gamma) \cdot \text{Gap}(\%), \tag{6}$$

where the balance weight γ determines the importance of the illegal rate in S. We visualize the weighted score of different algorithms in Figure 4 varying the balance rate from 0 to 1.

For the two terms $\{\gamma, (1-\gamma)\}$ in the Equation 6, $\frac{1}{10}$ to $\frac{10}{1}$ should be a reasonable scale range for ratio $\gamma/(1-\gamma)$, since the goal of algorithms solving TSPTW is to optimize both of two objectives. We highlight the range with dotted lines. In three different sizes of problems, MUSLA-adpat keeps the lowest score within the reasonable range. Although slightly worse than *Greedy-MT* in problem scale of $N = 100$, MUSLA outperforms other baselines in most cases. In previous learning-based works, the weighted score is commonly calculated as $S = \gamma\text{Timeout} + (1-\gamma)\text{Gap}$ with a fixed value of γ. We show the difference between timeout and illegal rate later.

On the Medium dataset, greedy solutions tend to have high legality and low optimality; as for RL algorithms, a trivial strategy such as *Greedy-LT* is easy to explore but may cause local-optimal issues. In our training procedure, we do observe that the RL policy maintains a similar performance as *Greedy-LT* for a long period. Although *AM* eventually converges to comparable results in problem sizes of $N = 20, 50$, it still shows an imbalance result within a large-scale case. The performance of *JAMPR* is even worse, as it does not improve the optimality much in all cases and totally fails in cases where $N = 100$. The imbalance between the two objectives



(a) MEDIUM, $N = 20$ (b) MEDIUM, $N = 50$ (c) MEDIUM, $N = 100$

(d) HARD, $N = 20$ (e) HARD, $N = 50$ (f) HARD, $N = 100$

Figure 4: Weighted score of different models. The dotted lines highlight reasonable ranges of γ.

shows the backwardness of RL algorithms. In contrast, the SL algorithm naturally avoids learning imbalance strategies by imitating high-quality expert datasets. In this way, SL methods learn towards a single objective and can improve both optimality and legality simultaneously.

As mentioned in Section 5, the training and evaluation instances in the HARD dataset are sampled from different distributions. Compared with other models, the illegal rate of *AM* increases with a large margin compared with it in MEDIUM, $N = 20, 50$. The incremented values are +77.66% and +39.60% respectively. The increases for MUSLA and MUSLA-adapt are reasonable, which shows the generalizability potential of our method.

On MEDIUM dataset with a problem size of $N = 20$, MUSLA shows a lower illegal rate but a higher total timeout compared with *AM*. Intuitively, minimizing the timeout should also minimize the timeout rate. However, there is a difference between the two objectives in fact. Replacing the legality indicator illegal rate by total timeout, TSPTW is relaxed as a soft-constrained problem, where solutions can tolerate minor timeouts. The experimental results in Table 1 also illustrate the difference between the two types of problems. This counter-intuitive result proves that our method better models the hard-constrained problem, rather than simply tuning the results on the original method.

When comparing LKH3 with MUSLA, it is seen that MUSLA only exhibits shorter solution times for larger problem sizes. Although MUSLA is not capable of completely replacing LKH, it serves as a feasible alternative. For scenarios where rapid response is prioritized, MUSLA presents a viable option, allowing for flexibility in the time-quality trade-off that real-world applications often necessitate.

6.3 Ablation Study

We conduct ablation experiments on a problem size of $N = 50$ to compare the effect of different components in our method. Corresponding to Section 4, we set up five different models: **Static** model is a trivial supervised learning model with only static information of TSPTW as input. **Dynamic** model adds dynamic information to improve the model's perception of the state of each step in the construction process following Section 4.1. **OSLA** introduces the one-step look-ahead mechanism for gathering information on constraint boundaries following Section 4.2.1. **MUSLA** and **MUSLA-adpat** are the multi-step look-ahead policy described in Section 4.2.2 and 4.3. It is clear that each part of the method improves the performance.

We also show the help of diverse supplement training datasets. **OSLA-MEDIUM** is an OLSA model trained using only MEDIUM dataset. The evident decline in performance indicates that diverse training datasets are necessary.

Compared with the heuristic method, learning-based policies still have a gap in performance. However, with learning-based methods, we can obtain solutions faster at the cost of a slight decrease in performance. According to the results in Table 1, MUSLA and MUSLA-adapt can achieve a 129x and 19x speedup separately using a single GPU at the problem size of $N = 100$. To further reduce the

solving time, OSLA, which has a reduced solution time and little performance degradation, may be a viable alternative.

7 CONCLUSION

In this paper, we propose a novel and effective solution for a challenging hard-constrained variant of TSP, TSPTW, named multi-step look-ahead (MUSLA). In particular, MUSLA is a supervised-learning method that adopts the looking-ahead information as the feature to improve the legality of TSP with Time Windows (TSPTW) solutions. To accurately evaluate and benchmark the statistical performance of various approaches, we also construct TSPTW datasets with hard constraints that can be used by the community to conduct follow-up research. With comprehensive experiments, MUSLA demonstrates great performance on diverse datasets, which is far better than existing baselines.

The limitation of our work lies in the requirement for expert datasets, which may be expensive to collect, especially in large-scale cases. It is the major difficulty that prevents us from trying larger-scale problems. In the future, we plan to improve the search strategy of MUSLA to collect more critical information while reducing the time-consuming. Utilizing suboptimal datasets generated by RL methods could also be a potential direction to address the data generation issues.

ACKNOWLEDGEMENTS

The Shanghai Jiao Tong University team is partially supported by Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102). The author, Jingxiao Chen, is supported by the Wu Wen Jun Honorary Doctoral Scholarship, AI Institute, Shanghai Jiao Tong University.

REFERENCES

[1] Majed Ghazi Alharbi, Ahmed Stohy, Mohammed Elhenawy, Mahmoud Masoud, and Hamiden Abd El-Wahed Khalifa. Solving traveling salesman problem with time windows using hybrid pointer networks with time features. *Sustainability*, 2021.
[2] David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde tsp solver. http://www.math.uwaterloo.ca/tsp/concorde, 2006. Accessed: 2023-05-18.
[3] Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016.
[4] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charlie Nash, Victoria Langston, Chris Dyer, Nicolas Manfred Otto Heess, Daan Wierstra, Pushmeet Kohli, Matthew M. Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *ArXiv*, abs/1806.01261, 2018.
[5] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Pr'emont-Schwarz, and André Augusto Ciré. Combining reinforcement learning and constraint programming for combinatorial optimization. *ArXiv*, abs/2006.01610, 2020.
[6] Jonas K. Falkner and Lars Schmidt-Thieme. Learning to solve vehicle routing problems with time windows through joint attention. *ArXiv*, abs/2006.09100, 2020.
[7] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. *ArXiv*, abs/2012.10658, 2020.
[8] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12, 2017.
[9] Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *ArXiv*, abs/1906.01227, 2019.
[10] Kamil Kaminski, Jan Ludwiczak, Maciej Jasiński, Adriana Bukala, Rafal Madaj, Krzysztof Szczepaniak, and Stanisław Dunin-Horkawicz. Rossmann-toolbox: a deep learning-based protocol for the prediction and design of cofactor specificity in rossmann fold proteins. *Briefings in Bioinformatics*, 23, 2021.
[11] Imdat Kara and Tusan Derya. Formulations for minimizing tour duration of the traveling salesman problem with time windows. *Procedia. Economics and finance*, 26:1026–1034, 2015.
[12] Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 35:1936–1949, 2022.
[13] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907, 2016.
[14] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.
[15] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.
[16] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017.
[17] Qiang Ma, Suwen Ge, Danyang He, Darshan D. Thaker, and Iddo Drori. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *ArXiv*, abs/1911.04936, 2019.
[18] Alex W. Nowak, Soledad Villar, Afonso S. Bandeira, and Joan Bruna. A note on learning algorithms for quadratic assignment with graph neural networks. *ArXiv*, abs/1706.07450, 2017.
[19] Christos Papalitsas, Konstantinos Giannakis, Theodore Andronikos, Dimitrios Theotokis, and Angelo Sifaleras. Initialization methods for the tsp with time windows using variable neighborhood search. *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–6, 2015.
[20] Emilio Parisotto, H. Francis Song, Jack W. Rae, Razvan Pascanu, Çaglar Gülçehre, Siddhant M. Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Manfred Otto Heess, and Raia Hadsell. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, 2019.
[21] Laurent Perron and Vincent Furnon. Or-tools. https://developers.google.com/optimization/, 2011. Accessed: 2023-05-18.
[22] Qiaoyue Tang, Yangzhe Kong, Lemeng Pan, and Choonmeng Lee. Learning to solve soft-constrained vehicle routing problems with lagrangian relaxation. *arXiv preprint arXiv:2207.09860*, 2022.
[23] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.
[24] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio', and Yoshua Bengio. Graph attention networks. *ArXiv*, abs/1710.10903, 2017.
[25] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NIPS*, 2015.
[26] Zhihao Xing and Shikui Tu. A graph neural network assisted monte carlo tree search approach to traveling salesman problem. *IEEE Access*, 8:108418–108428, 2020.
[27] Rongkai Zhang, Anatolii Prokhorchuk, and Justin Dauwels. Deep reinforcement learning for traveling salesman problem with time windows and rejections. *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.
[28] Jiongzhi Zheng, Kun He, Jianrong Zhou, Yan Jin, and Chumin Li. Reinforced lin-kernighan-helsgaun algorithms for the traveling salesman problems. *Knowl. Based Syst.*, 260:110144, 2022.

APPENDIX

A WEAKNESSES OF EXISTING DATASETS

This section demonstrates the weakness of TSPTW instances generated by previous learning-based papers. The coordination of TSPTW instances is sampled following uniform distribution, which is consistent with TSP papers. Only the generation of time windows is distinct. Alharbi *et al.* [1], Cappart *et al.* [5], Ma *et al.* [17] generate time windows following the visiting time t'_i of a constructed solution X'. The route X' is constructed following different methods.

Alharbi *et al.* [1], Ma *et al.* [17] construct route X' as a near-optimal TSP solution with learning-based and heuristic solver separately, which provides a too strong prior for the TSPTW instances. The solutions for generated TSPTW instances are basically in line with the solutions of TSP instances, *i.e.* TSPTW solvers can ignore the time window constraints. In order to verify our speculation, we generate a similar dataset, where route X' is constructed by greedily visiting the unvisited nearest neighbor. On this dataset, the **Greedy-MT** baseline reaches Gap = 0.00% and Illegal = 0.00%. Cappart *et al.* [5] generate X' with a random permutation, which seems a better choice. However, we test a greedy method that chooses the node i with the minimum earliest access time t^s_i, and the results are Gap = 0.00%, Illegal = 3.32%.

Zhang *et al.* [27] generate problem instances on a relaxed variant of TSPTW, called TSPTWR. The time windows are generated from a random distribution. However, the constraints are too tight for TSPTW. We generate problem instances following this paper and solve them with LKH3, only 1% instances can find solutions.

B DATASETS

In this paper, we generate TSPTW datasets with sizes of $n = 20, 50, 100$. For small problem sizes $n = 20, 50$, the training datasets consist of 500 000 problem instances. For big problem size $n = 100$, we only generated 50 000 TSPTW instances due to the long solution time of LKH3. The data volume ratio for the Medium, Hard, and supplementary training sets is 1 : 1 : 3. We generate Medium with hyperparameter of $\alpha = 0.5, \beta = 0.75$.

B.1 Hard Dataset

Training data. The training data of Hard samples time windows is based on the random distribution of Medium. For a TSPTW instance g with the size of n, we select $\lfloor 0.3n \rfloor$ nodes and divide them into n_g groups. Each group of nodes regenerates time windows based on the random distribution of Medium and adds an offset time on the time windows. More specifically, the generation process consists of the following four steps.

(1) Sample time windows following Medium dataset.
(2) Randomly pick $\lfloor 0.3n \rfloor$ nodes and divide them into k_p groups. For a problem size of $n = 20$, the total number of groups is $k_p = 2$. For problem sizes of $n = 50, 100$, the total number of groups is sampled from a range, $k_p \sim \mathcal{U}[2, 7]$.
(3) Individually resample time windows for each group following distribution of Medium. In particular, the parameter T_n for group with n_p nodes is $T_{n=n_p}$.
(4) For a group p, sample a start time t_p from a uniform distribution, $t_p \sim \mathcal{U}[0, T_n]$, then time windows of all nodes in this group is shifted with value t_p.

$$t^s_i \leftarrow t^s_i + t_p, \quad t^e_i \leftarrow t^e_i + t_p \tag{7}$$

Evaluation data. Evaluation Hard data are generated using the same method of training Hard data. The generation process of evaluation data is modified in steps 1 and 3. At step 1, time windows are constant values as $t^s = 0, t^e = T_n$. At step 3, time windows are constant values as $t^s = 0, t^e = T_{n=n_p}$.

B.2 Supplementary Training Datasets

Weakly constrained data. Medium and Hard datasets are designed to have conflicts in legality and optimality, which makes the learning-based policy are tends to generate legal solutions with poor performance. Therefore, we added two weakly constrained datasets based on Medium. The first dataset removes the constraint of earliest accessing time, *i.e.* $t^s_i = 0$. The second dataset removes both sides of time windows constraints, *i.e.* $t^s_i = t^e_i = 0$. For the second dataset, the TSPTW instances are relaxed to TSP instances.

Grouped Medium data. Grouped Medium data is designed as a supplementary of Medium and Hard. Similar to Hard training data, the grouped Medium data divided all n nodes into k_p groups. The shift value t_p of time windows in i-th group p is the maximum value of t^e in the (i-1)-th group, which ensures that time windows from two different groups do not cover each other.

C FEATURE DESIGN

C.1 Static Features

Static features are designed to describe specific TSPTW instance g. Problem instance g is a complete graph (V, E) consisting of the node set V and the edge set E. The properties of nodes are described with static node features, which are listed in Table 3. The properties of edges are described with static edge features and are listed in Table 4. In order to reduce the computational burden, we only retain the top 20% nearest neighbors to add edge features for each node.

C.2 Dynamic Features

Dynamic features are designed to describe all unvisited nodes $x' \in V \setminus X_{0:i}$ given a specific TSPTW instance g and current partial tour $X_{0:i}$. Dynamic features for node x' are listed in Table 5, where t_i is the current time for partial tour $X_{0:i}$.

C.3 Look-ahead Node Features

Following Section 4, we construct a look-ahead route X' for node x' and gather the look-ahead information I^d_{+1} or I^d_{+2} as look-ahead node features of x'. The look-ahead information I^d_{+1} and I^d_{+2} have the same feature design where only the way of constructing X' is different. The look-ahead features consist of two parts, constraint violation features and greedy features. For simplicity, we introduce the one-step look-ahead features. With partial route X', x' is the current node, and t_{i+1} is the current time.

The constraint violation features describe the possible delay caused by the current route X' for time constraint t^e. We denote the set of nodes that are already to be late as $X''_{\text{late}} = \{x'' \in V \setminus X' | t_{i+1} + L_{x',x''} > t^e_{x''}\}$. The detailed feature design is as follows.

Table 3: Static node features for node i.

Description	Feature	Dimension
node coordination	a_i	2
time windows	$\{t_i^s, t_i^e\}$	2
difference in coordinates between node i and starting node 0	$a_i - a_0$	2
distance to starting node 0	$L_{i,0}$	1

Table 4: Static edge features for edge (i, j).

Description	Feature	Dimension
distance between nodes i, j	$L_{i,j}$	1
difference in time windows between node i, j	$\{t_j^s - t_i^s, t_j^e - t_i^s, t_j^e - t_i^s, t_j^e - t_i^e\}$	4

Table 5: Dynamic features for node x' at step i of route construction.

Description	Feature	Dimension
node coordination	$a_{x'}$	2
difference in coordinates between node i and next node x'	$a_{x'} - a_{x_i}$	2
distance from current node x_i to next node x'	L_{x',x_i}	1
time spent visiting node x'	$\max(L_{x',x_i} + t_i, t_{x'}^s) - t_i$	1
time difference between time windows of node x' at step i	$\{t_{x'}^s - t_i, t_{x'}^e - t_i\}$	2
difference in time windows between node i, j	$\{t_{x'}^s - t_{x_i}^s, t_{x'}^e - t_{x_i}^s, t_{x'}^s - t_{x_i}^e, t_{x'}^e - t_{x_i}^e\}$	4

- Feature f_1^d denotes if the set X''_{late} is an empty set. If $X''_{\text{late}} \neq \emptyset$, route X' definitely cause a timeout.

$$f_1^d = \begin{cases} 1, & |X''_{\text{late}}| > 0 \\ 0, & |X''_{\text{late}}| = 0 \end{cases} \tag{8}$$

- Feature f_2^d, f_3^d represents the degree to which the current route X' violates constraints.

$$f_2^d = \max_{x'' \in X''_{\text{late}}} t_{i+1} + L_{x',x''} - t_{x''}^e \tag{9}$$

$$f_3^d = \sum_{x'' \in X''_{\text{late}}} t_{i+1} + L_{x',x''} - t_{x''}^e \tag{10}$$

For greedy features, we select the unvisited node x_g'' with the lowest time overhead, $x_g'' = \arg\min_{x'' \in V \setminus X'} \max(L_{x'',x'} + t_{i+1}, t_{x''}^s)$. The greedy features f_4^d, f_4^d are the distance and time overhead to node x_g''.

$$f_4^d = L_{x',x_g''} \tag{11}$$

$$f_5^d = \max(L_{x',x_g''} + t_{i+1}, t_{x_g''}^s) - t_{i+1} \tag{12}$$

In addition, the look-ahead information for some nodes is not gathered. For example, the visited nodes and nodes that are not searched in MUSLA do not have meaningful information I^d. We add an indicator $f_6^d = \{0, 1\}$ to indicate whether the look-ahead information of the node x' exists.

D EXPERIMENT HYPERPARAMETERS

Table 6 lists the common MUSLA parameters used in the experiments.

Received 19 August 2024; revised 19 August 2024; accepted 7 Oct. 2024

Table 6: MUSLA Hyperparameters

Parameter	Value
optimizer	AdamW [16]
number of hidden units per layer	128
number of hidden layers in e_d (MLP)	3
number of hidden layers in e_h(Transformer)	3
number of hidden layers in e_s (Graph Attention Network)	5
nonlinearity	ReLU
normalization	Layer Normalization [3]
learning rate	0.001
selection set of ϵ, $\mathcal{E}$	$\{-2, -1, -0.5, 0, 0.5, 1\}$