Deep Reinforcement Learning for Traveling Salesman Problem with Time Windows and Rejections

Rongkai Zhang*†✉, Anatolii Prokhorchuk*, Justin Dauwels*†
*School of Electrical and Electronic Engineering, †ST Engineering – NTU Corporate Lab
Nanyang Technological University
Singapore, Singapore
rongkai002@e.ntu.edu.sg✉, anatolii001@e.ntu.edu.sg, jdauwels@ntu.edu.sg

***Abstract*—Recently deep reinforcement learning has shown success in solving NP-hard combinatorial optimization problems such as traveling salesman problems, vehicle routing problems, job-shop scheduling problems, as well as their variants. However, most of the problems being solved are still relatively simple compared to the real-world scenarios. For instance, feasibility constraints are rarely considered in the current frameworks. This paper investigates the possibility of applying deep reinforcement learning to tackle combinatorial optimization problems with feasibility constraints. We propose a framework to solve such problems by combining deep reinforcement learning with a greedy heuristic. We demonstrate this approach for the traveling salesman problem with time windows and rejection (TSPTWR). The results show that our approach outperforms a commonly employed tabu search heuristic, both in terms of the solution quality and the inference computation time. More specifically, the inference process is 100 to 1000 times faster than tabu search for different size TSPTWR. The proposed approach can be considered as a framework enhancing reinforcement learning with heuristics for solving more complex problems.**

I. INTRODUCTION

Combinatorial optimization problems (COPs) are commonly found in many different domains, such as transportation, operations research, logistics, and telecommunications. The overall goal is to find an optimal solution, which can be modeled as a sequence of actions/decisions to maximize/minimize the objective function for a specific problem. However, once the action space becomes relatively large, the solution space can also dramatically increase leading to the intractability of finding the optimal solution.

Traditional approaches to tackling NP-hard graph optimization problems can be classified into three categories: exact algorithms, approximation algorithms, and heuristics. Exact algorithms are based on enumeration or branch-and-bound method with an integer programming formulation, that can guarantee the optimality. However, in general, they cannot be applied for large-scale problems. Polynomial-time approximation algorithms can (approximately) solve larger problems but may suffer from weak optimality guarantees. In addition, there are still many problems that do not have such approximations. Heuristics are commonly expressed as sets of rules for decision making, which are often fast, effective algorithms for NP-hard problems. However, they suffer from the lack of theoretical guarantees, the requirement of problem-specific knowledge, and manual trial-and-error design. Moreover, traditional approaches are instance-sensitive, which means that even though instances are sampled from the same distribution for the same problem, the algorithms will treat them as completely different problems and solve them again and again with no knowledge learned and shared.

With the development of deep neural networks (DNNs), much research on supervised DNNs for learning and representing heuristics has been conducted and yielded good performance. However, supervised learning requires a set of solutions to be known first to train the DNNs. A promising method to tackle this issue is Reinforcement Learning (RL). RL enables the algorithm to learn and evolve itself, either by interacting with an environment [1] or by inducing knowledge through a look-ahead search [2]. Recent research has shown that combining DNNs and RL, also known as deep reinforcement learning (DRL), is a powerful method for solving various types of COPs, such as traveling salesman problems (TSP) [3] [4] [5] [6] [7] [8], vehicle routing problems (VRP) [4], maximum cut (MC) [5] and so on.

Although current research has demonstrated exciting progress in solving COPs, those COPs are still relatively simple. For instance, most current work does not consider constraints. To the best of our knowledge, only recent work by Ma et al. [9] explicitly considers constrained versions of TSP. They propose to train a hierarchical graph pointer network (GPN) to solve TSP with time windows. We consider a modification of TSPTW that allows rejecting the nodes. This allows to model real-world applications when there is a need to balance the service levels with the service quality.

This paper proposes a framework to tackle TSPTWR, which is one typical COP with feasible constraints by enhancing DRL with simple heuristics. The framework can also be modified flexibly by replacing the heuristic for different problems. In Section 2, we give a more detailed review of DNNs and DRL for COPs. In Section 3, we define the problem and

The research was partially supported by the ST Engineering – NTU Corporate Lab through the NRF corporate lab@university scheme.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

introduce notations for TSPTWR. In Section 4, we illustrate the framework and the training process. In Section 5, we present our numerical results, whereas in Section 6, we discuss the results, and provide suggestions for future research.

II. RELATED WORK

The application of neural networks (NNs) for COPs dates back to 1985 when Hopfield and Tank applied a Hopfield-network for solving small TSP instances [10]. Recently, as deep learning became increasingly popular, many frameworks have been proposed for COPs.

Vinyals et al. [3] introduce the Pointer Network (Ptr-Net) as a model, which consists of a long-short term memory (LSTM) based encoder and an LSTM based decoder. The encoder encodes the input sequence to a higher dimensional representation, and the decoder uses attention as a pointer to select a member of the input sequence as the output, generating the solution step by step. The model is trained offline to solve TSP in a supervised learning manner. This work led to a trend of encoder-decoder based frameworks to represent and solve combinatorial optimization problems. To the best of our knowledge, most of the later frameworks follow this design type. However, there are two main limitations. The first one is the need for training data. The lack of good example solutions extremely limits this framework. The other is that the constraint on visiting each node exactly once may be violated.

Bello et al. [6] tackle the first issue by introducing a reinforcement learning method, namely the Actor-Critic algorithm, to train the Ptr-Net. They consider each instance as a training sample and use the tour length of a sampled solution for an unbiased Monte-Carlo estimate of the policy gradient. They address the second issue by recognizing that the constraint can be taken into account by simply masking the nodes already visited in the action space in the decoder.

Further, Nazari et al. [4] find that the input sequence should be irrelevant to the representation of an instance, so they replace the LSTM encoder by element-wise projections, such that the updated embeddings after state-changes can be effectively computed. They also apply this model to the various other COPs, such as the capacitated vehicle routing problem (CVRP).

Inspired by the Transformer framework [11], Deudon et al. [7] and Kool et al. [8] propose two self-attention based frameworks for TSP independently, which both showed improved performance. In [7] a deep reinforcement learning framework is combined with a heuristic, namely 2OPT. However, in this work, the heuristic is only utilized to improve the output sequence from the decoder. Essentially, the framework learns a good initial solution for 2OPT instead of the improved solution itself. In [8], a different decoder and a rollout training algorithm are proposed. The experiments show the performance is still good enough without 2OPT. Additionally, Kool et al. show that their framework can be modified to solve other types of COPs by changing the input, the mask, and the decoder context accordingly. However, as mentioned in the paper, such modification can be hard to design and interpret for some of the applications with feasibility constraints.

Instead of a separate encoder-decoder structure, Dai et al. [5] propose a model based on graph embeddings. They train the model to output the order in which nodes are inserted into a partial tour. Using a heuristic, the nodes are inserted at the best possible location greedily one by one. However, repeating embedding after each state-change is needed and the additional helper function also involves additional computations since it enumerates all the positions in each step.

A different approach is proposed by Wu et al. [12]. They propose to employ the DRL framework to learn the improvement heuristic for TSP. They train a self-attention architecture as a policy for selecting the next solution. Each action in their RL framework represents a node pair and the transition to the next state is calculated by a predetermined pairwise operator such as 2OPT.

We combine a self-attention based framework with heuristics as a whole framework. The advantages are threefold. First, the self-attention based framework embeds the graph only once. Second, the proposed framework is able to deal with the different constraints separately in either the self-attention framework or in the heuristic function. For example, in many variants of TSP or other COPs, a common constraint is that each node can be only visited once. We model such constraints by a mask in the decoder (as in [7]) and we employ heuristics for other constraints. For example, in TSPTWR, we accept and reject nodes according to the specified time windows. Hence, no manual design in the decoder context is needed as the constraints are explicitly dealt with by the heuristics. Third, the proposed framework can be applied to other combinatorial optimization problems by changing the heuristic. Since in the proposed approach the heuristic is inside the training loop, the reward is calculated from the solution following the heuristic function.

III. TSPTWR

A. Comparison of TSP and TSPTWR

Generally, TSP can be formulated as a fully connected graph problem. A problem instance s is denoted as a graph with n nodes, where node $i \in \{1, \ldots, n\}$ is represented by features x_i. x_i is the coordinate of node i, and the goal is to find the shortest tour π^* to visit all the nodes exactly once and return to the first node, namely the depot.

Hence, in TSPTWR, x_i is the 2-D coordinate of node i and the time window/deadline time of visiting node i. The goals are twofold, which are minimizing the rejection rate $R =$ (rejected nodes)/(total nodes) and meanwhile minimizing the total length of the tour L. That is when the rejection rates are the same, the solution with the shortest length is the optimal one, and vice versa. The comparison is shown in Table I.

TSPTWR is a more practical variant of TSPs, which is widely applied in logistics and scheduling, such as same-day delivery problems. It combines a common variant of TSP - TSPTW (Travelling Salesmen Problem with Time Windows, first mentioned in [13]) with the possibility of rejecting a

node. Carlton and Barnes [14] propose solving this problem via tabu search. In the proposed method the rejections are not considered explicitly but the penalty term consisting of the sum of all time windows violations is added to the cost function. In general, most of the literature on TSPTW focuses on various heuristics approaches. Gendreau et al. [15] propose an insertion heuristic to minimize travel times in TSPTW. Calvo [16] combines a heuristic approach for an assignment problem with a greedy insertion heuristic. Ohlmann and Thomas [17] propose a modification to a simulated annealing algorithm - compressed annealing. López-Ibáñez and Blum [18] describe a novel Beam-ACO method that combines ant colony optimization with beam search. TSPTWR can also be considered as a modification of the Prize Collecting Travelling Salesmen Problem (PCTSP, [19]). In PCTSP each node has associated prize and penalty values. The objective is to find a circuit that minimizes the travel cost plus the sum of penalties of unvisited nodes subject to collecting the minimum required prize. For an overview of this and related TSP problems, the reader is referred to [20]. The solution space of TSPTWR is much larger

TABLE I
COMPARISON OF TSP AND TSPTWR.

	Features	Constraints	Cost
TSP	2D coordinates	Visit time	Tour length
TSP TWR	2D coordinates + Time window	Visit time +Time window	Tour length +Rejection rate

than the one of TSP due to the possibility of rejections. For an instance with n nodes, the number of possible solutions of TSP S_{TSP} is half of all permutation of n, i.e., $\frac{n!}{2}$. For TSPTWR the size of the space is even larger. In case of the largest TSPTWR where all nodes can be visited, since rejections can be made, the solution space consists not only of permutations of n cities, but of permutations of all subsets of n cities. The cardinality of the solution space is then given by:

$$S_{\text{TSPTWR}} = \frac{n!}{2(n-1)!} + \sum_{i=1}^{n} \frac{n!}{2(n-i)!}. \tag{1}$$

B. TSPTWR as a Sequential Decision Making Problem

TSPTWR can be modeled as a sequential decision making problem by generating the final solution consecutively based on the partial solution. Given an instance s with n nodes, at step i decision a_i is made based on the previous decisions, which is a partial solution $\pi_{i-1} = \{a_1, a_2, ..., a_{i-1}\}$. In TSPTWR, a decision can be either picking a node to visit or rejecting a node. The policy for decision making can be modeled by a DNN with parameter Θ. Hence the probability of a_i is modeled by $P_\Theta(a_i|\pi_{i-1}, s)$ and the probability $P_\Theta(\pi)$ of a completed tour π is formulated as (2):

$$\begin{aligned} P_\Theta(\pi|s) &= P_\Theta(a_1|s)P_\Theta(a_2|a_1,s)P_\Theta(a_3|a_3,a_2,a_1,s) \\ &\quad ...P_\Theta(a_n|a_{n-1}...,a_1,s) \\ &= \prod_{i=1}^{n} P_\Theta(a_i|\pi_{i-1},s). \end{aligned} \tag{2}$$

Eventually, a well-trained DNN with the optimal parameter Θ should make $P_\Theta(\pi^*)$ as large as possible. There are two ways to obtain π^*. The first one is following the literature to design an end-to-end framework that can accept or reject a node at each step until termination. Let us consider a situation when a rejection should be made. The first case is when it is not possible to serve node x within the deadline, hence it should be rejected. The second case occurs when accepting a node x will cause missing other requests, and therefore, increasing the overall rejection rate. The first one can be achieved by a simple mask in action space. However, for the second situation, the DNN needs a new decoder context or an even more complex mechanism to figure out if such rejection should be made. To overcome the difficulty in the design, we propose a different approach, that is enhancing the DNN for TSP with a simple heuristic helper to solve TSPTWR. First, the DNN will output a solution for TSP and a greedy helper function will reject the nodes that cannot be visited and give the reward based on the rejection rate and the tour length. This reward is then used for training the DNN for TSPTWR.

IV. OUR FRAMEWORK

A. Overall Framework

Our framework also follows the self-attention based encoder-decoder perspective [8], capable of solving general TSP. However, the proposed framework differs from it by the addition of an explainable heuristic helper function that post-processes the output sequence from the decoder. The helper function allows the framework to assess the solution quality (reward) correctly according to the setting of TSPTWR. The diagram of the proposed framework is shown in Fig 1. Algorithm 1 shows the process of obtaining the optimal parameters Θ^* for the DNN and each part is described in detail next.

B. Encoder

From the d_x-dimensional input features x_i, the encoder computes initial d_h-dimensional node embeddings $h_i^{(0)}$ through a learned linear projection with parameters W_x and b_x : $h^i_{(0)} = W_x x_i + b_x$. For TSPTWR with deadline time constraints, d_x = 3, where the features are 2-D coordinate and the deadline time t_i of each node. For TSPTWR with two-side time windows, d_x = 4, where the features are 2-D coordinate and time windows (t_i^{start}, t_i^{end}). Here, we consider d_x = 3 to illustrate the framework. The embeddings are updated through N attention layers, each consisting of two sublayers to $h^i_{(N)}$. The two sublayers are a multi-head attention (MHA) layer that executes message passing between the nodes and a node-wise fully connected feed-forward (FF) layer. Each sublayer adds a skip-connection and batch normalization. The encoder computes an aggregated embedding $\bar{h}_{(N)}$ of the input graph as the mean of the final node embeddings. Both the node embeddings $h^i_{(N)}$ and the graph embedding $\bar{h}_{(N)}$ are used as input to the decoder. Algorithm 2 shows the whole process.

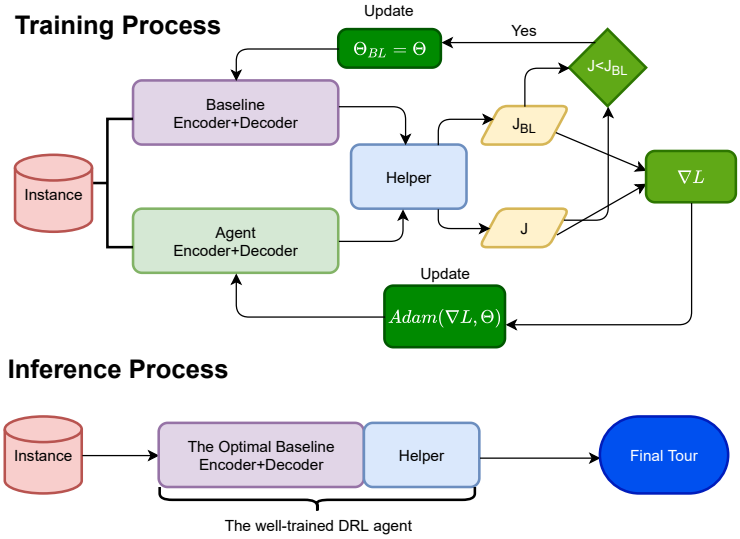

Fig. 1. The diagram of our framework.

Algorithm 1 The Whole Framework for TSPTWR

Input: Instances S, Batch Size S_{bs}, Training Epoch E, Update Threshold $\alpha > 0$

Initialize Θ, Θ^{BL}
for epoch = 1:E **do**
  for all S **do**
    Sample S_{bs} instances from S
    for $S_i \in S_{bs}$ **do**
      $H_i = Encoder_{\Theta}(S_i)$,$H_{BLi} = Encoder_{\Theta^{BL}}(S_i)$
      $P_i = Decoder_{\Theta}(H_i)$,$P_{BLi} = Decoder_{\Theta^{BL}}(H_i)$
      $\pi_i = Sample(S_i, P_i)$
      $\pi_{BLi} = GreedySample(S_i, P_{BLi})$
      $J_i = Helper(\pi_i)$
      $J_{BLi} = Helper(\pi_{BLi})$
    end for
    $\nabla L \propto \sum_1^{S_{bs}} (J_i - J_{BLi}) \nabla_{\Theta} log P_i(\pi_i)$
    $\Theta = Adam(\Theta, \nabla L)$
    if $J_i - J_{BLi} < -\alpha$ **then**
      $\Theta^{BL} = \Theta_i$
    end if
  end for
end for
Output $\Theta^* = \Theta^{BL}$

Algorithm 2 Encoder

Input: Features x_i, $i = (i, ..., n) \in s$, Normalization Constant d_k, Layers Number N

Embed to high dimension: $h^i_{(0)} = W_x x_i + b_x$
for l = 1:N **do**
  Compute the key k_i, value v_i and query q_i for each node: $k_i = W^K h^i_{l-1}$, $v_i = W^V h^i_{l-1}$, $q_i = W^Q h^i_{l-1}$
  Compute the compatibilitires: $u_{ij} = \frac{q_i^T k_j}{\sqrt{d_k}}$, $i \neq j$
  Compute the attention weights using a softmax: $a_{ij} = \frac{e^{u_{ij}}}{\sum_j e^{u_{ij}}}$
  Output from the MHA sublayer: $h^{i'}_l = \sum_j a_{ij} v_j$
  Output from FF sublayer: $h^{i''}_l = W^{ff_1} ReLu(W^{ff_0} h^i_{l-1} + b^{ff_0}) + b^{ff_1}$
  Combine and Batch Normalize: $h^i_l = BN(h^{i'}_l + h^{i''}_l)$
end for
Compute the aggregated embedding $\bar{h}_{(N)} = \frac{\sum_i h^i_l}{n}$
Final embedding $h^i_{(N)}$, $\bar{h}_{(N)}$

C. Decoder

The decoder selects one non-visited node at time $t \in (1, ..., n)$ based on the decode context, which is the overall embedding $\bar{h}_{(N)}$ and the visited nodes (partial tour) π_t. Following the same design as in [8], we consider the first node and the last node in the partial tour as a representation of π_t. The decode context is obtained by the same encoder shown previously. The selection is realized by encoding the decoded context to a query and comparing it with the key of each non-visited node. Then softmax is applied to compute the probability of choosing each node, and according to the probability, one node is added to the tour π sequentially. The constraints on only visiting each node exactly once are guaranteed by masking the visited node with zero probability to be selected. Algorithm 3 summarizes the different steps.

D. Helper Function

After obtaining the total tour π, we should consider the constraints on the time windows/deadline in order to obtain the final solution for TSPTWR. From the previous discussion, we conclude that the rejections should be made in two situations. Here, we argue that both of them can be learned by a greedy heuristical helper function. The helper function checks the feasibility of each node sequentially. If a node violates the

Algorithm 3 Decoder

Input: Overall embedding $\bar{h}_{(N)}$, Key for nodes K, Mask

Initialize two trainable place holders for step 0: v_f, v_l

Compute initial decode context: $q_d = W^{Q_d} Concate(Encoder(\bar{h}_{(N)}, v_f, v_l))$

for i = 1:n **do**

  Compute the compatibilities: $u_{di} = \frac{q_d^T k_j}{\sqrt{d_k}}$, $i = (1, ..., n)$

  Mask the visited nodes: $u_{di} = -\infty$, $i \in \pi_{t=i}$

  Compute probability of visiting node i: $p_{di} = \frac{e^{u_{di}}}{\sum_i e^{u_{di}}}$

  Sample a node from the probability: $\pi_i = Sample/GreedySample(h^i_{(N)}, p_{di})$

  Update v_f,v_l, Mask: $v_f = \pi_1$,$v_f = \pi_i$

  Update decode context : $q_d = W^{Q_d} Concate(Encoder(\bar{h}_{(N)}, v_f, v_l))$

end for

Solution for TSP: π

deadline, the heuristic will reject it and check the following nodes. Algorithm 4 shows how our helper function enhances the DRL framework to solve TSPTWR.

Algorithm 4 Helper function

Input: Solution for TSP π, Instances S

Initialize time $t = 0$,

for i = 0:$n-1 \in \pi$ **do**

  $t = t + t_{i,i+1}$

  where $t_{i,i+1}$ is the time needed from x_i to x_{i+1}

  if $t > t_{i+1}$ **then**

    $t = t - t_{i,i+1}$

    delete x_{i+1} in π

  end if

end for

Solution for TSPTWR: π_{TSPTWR}

The reward/cost J of a solution is calculated according to π_{TSPTWR}:

$$J = C * \text{RejectionRate} + \text{TourLength}(\pi_{\text{TSPTWR}}), \quad (3)$$

where C is a tuning weight that controls the importance of rejection. The weight C should be large enough to force the framework not to learn tricks, such as reducing J by serving no nodes and maintain a high level of service.

E. Training Method

To train our framework, we define the loss $L(s) = E_{p_\theta(\pi|s)}[J(\pi) - J_{BL}(\pi)]$, which is the expectation of the cost given an instance s based on the parameter Θ in the framework. By applying the policy gradient-based training REINFORCE with baseline [21], [22], we can adjust Θ by Monte-Carlo sampling to minimize the loss [23]:

$$\nabla L \propto \sum_{i=1}^{S_{bs}} [(J_\Theta(\pi_i) - J_{BL}(\pi_{BLi})] \nabla_\Theta \log p_\Theta(\pi_i), \quad (4)$$

where $J_{BL}(\pi)$ is the cost of a greedy rollout sampled from the baseline framework. The baseline framework has the same architecture as the agent but has different parameter Θ^{BL}. During the training, the parameter Θ^{BL} is updated to Θ, if J_θ is less than J_{BL}. This baseline is not only exploited to adjust Θ, but it is also able to reinforce the good solutions by increasing the probability. The final parameter Θ^* for inference is set to Θ^{BL}. The loss function L and its derivative are computed as follows:

$$\begin{aligned} L(s) &= E_{p_\theta(\pi|s)}[J(\pi)] = \sum_\pi p_\Theta(\pi|s)[J_\Theta(s,\pi) - J_{BL}(s,\pi)]. \\ \nabla L &= \sum_\pi \nabla_\Theta p_\Theta(\pi|s)[(J_\Theta(\pi) - J_{BL}(\pi_{BL})] \\ &= \sum_\pi p_\Theta(\pi|s) \nabla_\Theta \log p_\Theta(\pi|s)[(J_\Theta(\pi) - J_{BL}(\pi_{BL}))] \\ &\propto \sum_{i=1}^{S_{bs}} [(J_\Theta(\pi_i) - J_{BL}(\pi_{BLi})] \nabla_\Theta \log p_\Theta(\pi_i). \end{aligned}$$

V. EXPERIMENTS AND RESULTS

In the numerical experiments, we consider two types of TSPTWR problems. The first one is with two-side time windows (t_i^{start}, t_i^{end}). A node i can be visited before t_i^{end}, but if the agent visits the node before t_i^{start}, it should wait until t_i^{start} to leave. The second one is only with the deadline time constraints, which can be considered as a special case of the two-side constraints with all $t_i^{start} = 0$. The agent can visit a node before its deadline and leave without waiting, which is also known as the same-day delivery problem.

A. Data Generation

We generate the training instances by sampling n nodes uniformly at random in the unit square and setting the depot in the center (0.5,0.5). TSPTWRn indicates that the size of TSPTWR is n. For the first type, we consider TSPTWR30 and TSPTWR50. According to the problem size, the start time t_i^{start} is randomly generated from different uniform distributions U and time window TW also varies. These experiments aim to validate our DRL framework and also investigate the influence of different weight value C. For the second type, the start time is set as zero and the deadline time for each node is sampled from different uniform distributions U. Larger problems are considered to validate the scalability of our framework. Table II summarises the setting of all the experiments.

TABLE II
EXPERIMENT SETTINGS.

n	Two-side Time Window		Deadline Constraints	
	Start time	TW	Start time	Deadline time
30	U=[0,3]	1,2,3	0	U=[0,3]
50	U=[0,5]	1,2,5	-	-
100	-	-	0	U=[0,5]
150	-	-	0	U=[0,10]

We set the distance between two nodes equal to the travel time. Examples of instances with 4 nodes are shown in Fig. 2. The test instances are generated in the same way but with different random seeds. For each setting, we generate 1000 instances. The performance is the overall cost and/or rejection rate.

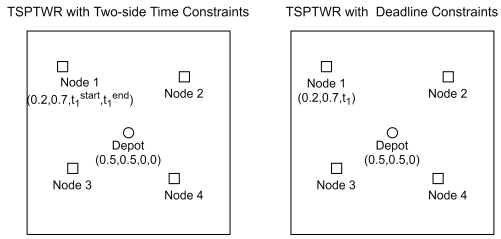

Fig. 2. Instances with 4 nodes for Two Different Settings.

B. Setting of Baseline and Our Framework

As a baseline we consider tabu search [24], which is a well-known meta-heuristic and outperforms others for large scale COPs. Since tabu search is sensitive to the random initial solution and usually costs more time, we sample five instances in each setting, apply tabu search to solve a test instance 5 times and average the cost, the rejection rate, and the computation time. The hyper-parameters in tabu search for TSPTWR with n nodes are presented in Table III.

TABLE III
HYPERPARAMETERS FOR TABU SEARCH.

Number of Total Possible Actions	N_A
Tabu Length TL	$0.5N_A$
Maximum Iterations It	200
Termination Threshold β	$1*10^{-6}$

The solutions can be manipulated by swapping, reversing, and inserting, leading to a total number N_A of possible actions given by:

$$N_A = \frac{n(n-1)+(n-2)(n-3)}{2} + (n-1)(n-2). \quad (5)$$

The tabu length is selected as $0.5N_A$ to balance the performance and the computation time [25]. The whole search terminates in 200 iterations or when the improvement is smaller than 10^{-6}.

Table IV shows the hyper-parameters for DRL training.

TABLE IV
HYPERPARAMETERS FOR DRL.

Training Dataset Size S	10240
Random Seed for Training	1
Batch Size S_{bs}	128
Training Epoch E	150
Learning Rate	$1*10^{-4}$
Layers Number N	3
Weight C for two-side constraints	0.1 1 10 100
Weight C for TSPTWR30 with deadline constraints	10
Weight C for TSPTWR100 with deadline constraints	20
Weight C for TSPTWR150 with deadline constraints	30

C. Results

We investigate how different weight values C influence the rejection rate in problems with two-sided time windows. Table V shows that increasing C will reduce the rejection rate, which is in agreement with our design intuition. The rejection rate obtained by tabu search, the baseline method, is also given for comparison. Moreover, we conduct more experiments with different time windows and weight values C. Fig. 3 illustrates the rejection rate for problems with different combinations of time windows and weight C. The results also reveal that the rejection rate can be controlled by tuning C. Based on

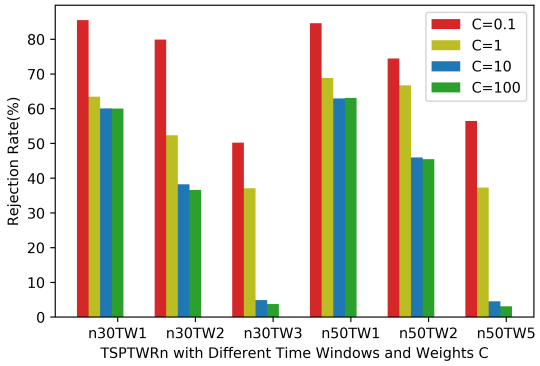

Fig. 3. Rejection rate for TSPTWR with different settings.

our findings, we select different weights C for TSPTWR with deadline constraints. Fig. 4 illustrates the learning curve for TSPTWR100 with deadline constraints, where one training step is equal to one batch. It shows that our framework can learn and converge to a good policy to reduce the average cost after sufficient learning steps. The training time for each epoch increases with the size of TSPTWR, but thanks to the parallelization of self-attention, for the largest size we consider here, it still takes less than three minutes. Table VI shows the performance of DRL and tabu search for TSPTWR with deadline constraints.

In addition to evaluating the framework on randomly generated data, we investigate the performance of the model on some popular instances for TSPTW. Specifically, we perform tests on a subset of instances from Potvin and Bengio [26].

TABLE V
INFLUENCE OF DIFFERENT C VALUES ON REJECTION RATE IN DRL AND TABU SEARCH FOR TSPTWR WITH TIME WINDOWS.

	C	DRL: Rejection Rate(%)	Tabu: Rejection Rate(%)
n=30, TW=3	0.1	50.23	**45.09**
	1	37.09	**35.62**
	10	**4.89**	11.76
	100	**3.73**	4.81
n=50, TW=5	0.1	56.45	**54.53**
	1	**37.28**	38.93
	10	**4.52**	15.64
	100	**3.11**	5.17

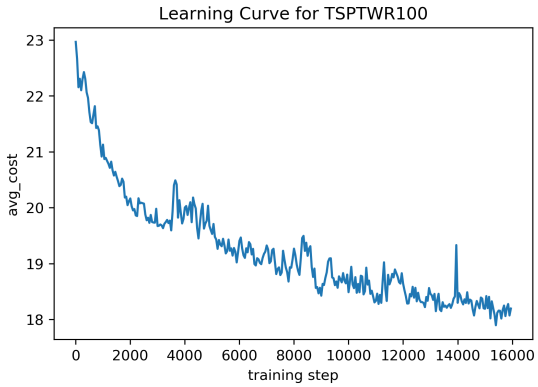

Fig. 4. The learning curve for TSPTWR100.

These instances are based on Solomon's RC2 instances [27], which contain different numbers of customers sampled from a mix of clustered and uniform distribution in a 100x100 area. The ready times are uniform $U = [0, 900]$, time window length is fixed at 120 and the service time is fixed at 10. We generate the training data with the following parameters: 25 customers, customer coordinates are distributed uniformly $U = [0, 100]$, the depot is located at (40,50), and the others remain the same. This results in training data being relatively similar to the test instances. At the same time, it provides an opportunity to investigate the generalization ability of the framework, given the differences in location distribution. It should also be noted, that in TSPTW the objective is to find a tour in the graph, while the trained model is still allowed to reject some of the nodes. To prohibit rejections as much as possible, the penalty factor C is set to 100000 during the training. Table VII shows the best-known value of the makespan objective for each of the test instances and the results obtained by the proposed DRL approach. It can be observed that for some of the instances (rc201.1) our method achieves the same results as the best known.

VI. DISCUSSION AND FUTURE WORK

Based on the results, it can be observed that the proposed DRL based framework can learn a relatively good heuristic for solving different instances within a reasonable time as long as the distribution of instances is known. The computation time of the tabu search increases significantly with the size of TSPTWR. By contrast, since the inference with DRL is a direct mapping, the computation time increases only slightly with the dimensionality of the problem. Fig. 5 shows the computation time for inferences for different sizes of TSPTWR for both DRL and tabu search. The results also show that

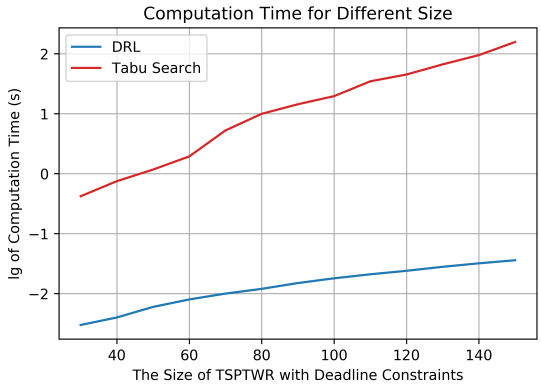

Fig. 5. Computation time for different size TSPTWR.

our DRL framework achieves some robustness judging by the low variance. It should also be noted that even with the similar overall costs between the proposed method and the tabu search, the rejection rate for the DRL framework is lower. This can be achieved by the multi-head attention mechanism in the encoder, because it allows nodes to communicate relevant information over different channels, such that the node embeddings include valuable information about the node in the context of the graph. Such additional information helps our framework to make better decisions than just randomly searching, as in tabu search.

The proposed method can be seen as an exploration of how to solve combinatorial optimization problems with constraints via deep reinforcement learning with heuristics. The general principle behind it is intuitive. We proposed to embed some rules into the network architecture, and then employ a known heuristic for other constraints. Such a divide-and-conquer approach can greatly simplify the complexity of architecture design and improve the interpretability of the model. Moreover, this method can be adapted for different problems by replacing the helper (heuristic) function accordingly.

Right now we only consider one agent and one type of constraint (time windows). In future work, we will consider several directions. The first one is to consider more agents and the cooperation between them, which is known as a multi traveling salesman problem (mTSP). It can potentially be solved by applying the proposed framework to solve one mTSP greedily and iteratively until all the feasible requests are served. A different direction is to consider more complex problems with more constraints, such as capacitated vehicle routing problems with time window and rejections (CVRPTWR) or dial-a-ride-problem (DARP).

TABLE VI
COMPARISON OF THE PERFORMANCE OF THE PROPOSED DRL FRAMEWORK AND TABU SEARCH ON TSPTWR WITH DEADLINES.

n	DRL				Tabu			
	cost	tour length	rejection rate(%)	time(s)	cost	tour length	rejection rate(%)	time(s)
30	8.48 ± 0.03	2.79	**56.86**	0.003	**8.24±0.44**	1.58	66.66	0.48
	8.51±0.03	2.79	**57.21**	0.003	8.56±0.17	1.89	66.66	0.38
	8.47±0.03	2.78	**56.94**	0.003	8.55±0.56	2.82	57.33	0.41
100	**17.85±0.04**	4.95	64.52	0.021	17.85±1.04	5.01	**64.20**	20.70
	17.88±0.04	4.95	**64.64**	0.021	18.13±0.89	2.93	76.00	18.46
	17.87±0.04	4.94	**64.63**	0.020	17.98±0.66	2.82	75.80	19.70
150	**24.25±0.06**	8.51	**52.48**	0.036	24.27±3.30	5.63	62.13	167.85
	24.25±0.06	8.52	**52.43**	0.036	24.33±4.40	4.89	64.79	150.49
	24.22±0.06	8.49	**52.42**	0.035	24.48±1.64	4.48	66.66	151.30

TABLE VII
RESULTS FOR INSTANCES FROM POTVIN AND BENGIO [26]

Instance	n	Best known	DRL makespan	DRL rejection rate(%)
rc201.1	19	592.06	592.06	0
rc201.2	25	860.17	865.51	8.00
rc201.3	31	853.71	853.71	19.35
rc201.4	25	889.18	919.23	12.00

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge Ramasamy Pandi Ramesh, Tian Mengxuan, Ye Xiaohong and Zhang Cong for the helpful discussions and proofreading.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–, 2017.

[3] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 2692–2700.

[4] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takac, "Reinforcement learning for solving the vehicle routing problem," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 9839–9849.

[5] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 6348–6358.

[6] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*, 2017.

[7] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, "Learning heuristics for the tsp by policy gradient," in *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2018, pp. 170–181.

[8] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations*, 2019.

[9] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori, "Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning," *arXiv preprint arXiv:1911.04936*, 2019.

[10] J. J. Hopfield and D. W. Tank, ""neural" computation of decisions in optimization problems," *Biological cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[11] 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 30*, 2017, pp. 5998–6008.

[12] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving the travelling salesman problem," *arXiv preprint arXiv:1912.05784*, 2019.

[13] E. K. Baker, "An exact algorithm for the time-constrained traveling salesman problem," *Operations Research*, vol. 31, no. 5, pp. 938–945, 1983.

[14] W. B. Carlton and J. W. Barnes, "Solving the traveling-salesman problem with time windows using tabu search," *IIE transactions*, vol. 28, no. 8, pp. 617–629, 1996.

[15] M. Gendreau, A. Hertz, G. Laporte, and M. Stan, "A generalized insertion heuristic for the traveling salesman problem with time windows," *Operations Research*, vol. 46, no. 3, pp. 330–335, 1998.

[16] R. W. Calvo, "A new heuristic for the traveling salesman problem with time windows," *Transportation Science*, vol. 34, no. 1, pp. 113–124, 2000.

[17] J. W. Ohlmann and B. W. Thomas, "A compressed-annealing heuristic for the traveling salesman problem with time windows," *INFORMS Journal on Computing*, vol. 19, no. 1, pp. 80–90, 2007.

[18] M. López-Ibáñez and C. Blum, "Beam-aco for the travelling salesman problem with time windows," *Computers & Operations Research*, vol. 37, no. 9, pp. 1570–1583, 2010.

[19] E. Balas, "The prize collecting traveling salesman problem," *Networks*, vol. 19, no. 6, pp. 621–636, 1989.

[20] D. Feillet, P. Dejax, and M. Gendreau, "Traveling salesman problems with profits," *Transportation science*, vol. 39, no. 2, pp. 188–205, 2005.

[21] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[22] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12*, 2000, pp. 1057–1063.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[24] M. Zachariasen and M. Dam, "Tabu search on the geometric traveling salesman problem," in *Meta-Heuristics*. Springer, 1996, pp. 571–587.

[25] F. Glover and E. Taillard, "A user's guide to tabu search," *Annals of operations research*, vol. 41, no. 1, pp. 1–28, 1993.

[26] J.-Y. Potvin and S. Bengio, "The vehicle routing problem with time windows part ii: genetic search," *INFORMS journal on Computing*, vol. 8, no. 2, pp. 165–172, 1996.

[27] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations research*, vol. 35, no. 2, pp. 254–265, 1987.