# Learn to Solve the Min-max Multiple Traveling Salesmen Problem with Reinforcement Learning

Junyoung Park
KAIST
Daejeon, South Korea
junyoungpark@kaist.ac.kr

Changhyun Kwon
University of South Florida
Tampa, United States
chkwon@usf.edu

Jinkyoo Park
KAIST
Daejeon, South Korea
jinkyoo.park@kaist.ac.kr

## ABSTRACT

We propose ScheduleNet, a scalable scheduler that minimizes task completion time by coordinating multiple agents. We formulate the min-max Multiple Traveling Salesmen Problem (mTSP) as a Markov decision process with an episodic reward and derive a scalable decision-making policy using Reinforcement Learning (RL). The decision-making procedure of ScheduleNet includes (1) representing the state of a problem with the agent-task graph, (2) extracting node embedding for agents and tasks by employing the type-aware graph attention, (3) and computing the task assignment probability with the computed node embedding. We show that ScheduleNet can outperform other heuristic approaches and existing deep RL approaches, particularly validating its exceptional effectiveness in solving large and practical problems. We also confirm that ScheduleNet can effectively solve practical mTSP variants, which include limited observation and online mTSP.

## KEYWORDS

Minmax Nultiple Traveling Salesmen Problem (mTSP); Routing; Scheduling; Graph Neural Network; Reinforcement Learning

**ACM Reference Format:**
Junyoung Park, Changhyun Kwon, and Jinkyoo Park. 2023. Learn to Solve the Min-max Multiple Traveling Salesmen Problem with Reinforcement Learning. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

## 1 INTRODUCTION

As an important extension of the classical Traveling Salesman Problem (TSP), the multiple TSP (mTSP) arises in various scheduling, routing, and planning problems. The applications of mTSP include print press scheduling [12], crew scheduling [22, 35], interview scheduling [11], hot rolling scheduling [36], robot/drone routing [2], and disaster management [4, 7, 26]. When there is only one salesman, mTSP is identical to TSP.

This paper considers the mTSP with the min-max objective rather than the min-sum objective. While the min-sum objective minimizes the total sum of each salesman's cost, the min-max objective minimizes the highest cost among all salesmen. Hence, the min-max objective is more appropriate when the underlying application is a scheduling problem, as it minimizes the total completion time or the makespan.

*Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.* 

This paper proposes an approximate method based on reinforcement learning (RL) to solve large-scale instances of the min-max mTSP. However, solving a large-scale mTSP using mathematical programming becomes infeasible or inefficient due to the expensive computational cost. Moreover, such approaches are less applicable in solving real-time scheduling problems where the new tasks appear dynamically.

Related to mTSP, the Capacitated Vehicle Routing Problem (CVRP) also extends TSP when each salesman (equivalently, a vehicle) has a finite capacity to visit customers. On the other hand, when the capacity is unlimited, then CVRP is reduced to mTSP. Therefore, most RL approaches developed for CVRP are applicable to solve mTSP with the min-sum objective.

The min-max objective, however, has received little attention in the RL literature—unlike the min-sum objective in TSP and CVRP [5, 20, 21]—due to the additional complexity it introduces. While CVRP with the min-sum objective can be solved by routing a single vehicle multiple times *sequentially*, the min-max objective cannot be handled similarly since minimizing the makespan requires *coordination* among salesmen. Moreover, the finite capacity of each vehicle in CVRP plays a crucial role in the existing RL-based approaches to result in multiple routes; hence applying such an approach to the min-max mTSP is inappropriate.

To handle the distinct challenges of developing an RL approach for the min-max mTSP, we propose a scalable policy, ScheduleNet. Our approach allows each agent to choose its destination independently, using local observations and incorporating other agents' assignments. This ensures that the learned policy can solve a large-scale problem without searching over the joint action space for all agents. However, a sophisticated coordination mechanism should be incorporated to make the independently-chosen scheduling decision produce an excellent global performance.

**Decision-Making Scheme.** We formulate the min-max mTSP as a Markov Decision Process (MDP) with an episodic reward and derive a decision-making policy using RL. At every step of the MDP, ScheduleNet accepts the MDP state as an input and assigns an idle agent to one of the feasible tasks. The decision-making procedure of ScheduleNet is as follows:

- ScheduleNet first represents the MDP state as an agent-task graph, effectively capturing the complex relationships among the entities to be applied in mTSP.
- ScheduleNet employs the Type-aware Graph Attention (TGA) to extract important relational features among the agents and tasks for the best cooperative task assignment.
- ScheduleNet computes the agent-task assignment probability using the computed node embeddings.

**Training Method.** Although the makespan (shared team reward) is the most direct and general reward design for solving mTSP, training a scheduling policy using this reward is difficult due to the credit assignment issues [13, 31]. Additionally, makespan is highly volatile due to the combinatorial aspect of mTSP solution space (i.e., a slight change in the solution can drastically alter the outcome). To overcome these issues, we employ the Clip-REINFORCE algorithm with normalized reward to train the cooperative policy effectively.

**Novelties.** The proposed method that derives the constructive scheduling policy to coordinate multiple agents has the following novelties and advantages:

- ScheduleNet extracts crucial features using TGA and efficiently assigns the best cooperative task. Furthermore, the computationally-efficient representation scheme and the scalable decision-making scheme allow ScheduleNet to solve large-scale instances of mTSP.
- The agent-task graph representation and TGA allow the trained policy to solve problems with different numbers of agents and tasks while showing better generalization capabilities compared to the neural baselines.
- ScheduleNet can solve practical variants of mTSP, including the limited communication scenario where each agent (i.e., salesman) decides the actions (i.e., visiting cities) based on the local observation, and the online scenario where the cities dynamically appear during the execution of the scheduled actions.

## 2 RELATED WORK

**RL approaches that solve vehicle routing problems**. According to Mazyavkina et al. [25], the RL approaches that solve vehicle routing problems can be categorized into: (1) the improvement heuristics that rewrite the complete solution iteratively to obtain a better routing plan [8, 9, 18, 24, 39], (2) the construction heuristics that construct the solution sequentially by assigning idle vehicles to unvisited cities until the complete routing plan (sequence) is constructed [5, 17, 20, 27], and (3) the hybrid approaches that blend both approaches [1, 10, 16, 19]. Typically, the improvement heuristics show better performances than construction heuristics as they revise the complete plan iteratively. These RL approaches have exclusively focused on static planning in a single-agent perspective, which is far from the settings of real applications; however, the construction heuristics are more effective for online vehicle routing problems, where the routes should be updated whenever a new customer appears.

**RL approaches that solve the min-max mTSP**. There are only a few RL approaches that solve the min-max mTSP, which involves minimizing the makespan for multiple salesmen to visit all cities. [15] applies RL to train the clustering algorithm that groups cities, and strong TSP heuristics (e.g., OR-Tool) to optimize the sub-tours within each city cluster. This is fundamentally different from ScheduleNet, which derives a complete end-to-end learned heuristic that constructs a feasible solution from "scratch" without relying on any existing solvers. [6] proposes a transformer-based construction policy to solve the min-max mTSP, is the most similar approach to ScheduleNet; however, ScheduleNet shows better scheduling performance and robustness to the input distribution shift compared to [6].

## 3 PROBLEM FORMULATION

We formulate the min-max mTSP as an MDP with sparse reward and aim to derive a scalable scheduling policy that can be shared by all agents. Let us consider the single-depot mTSP with two types of entities, $m$ salesmen (i.e., $m$ agents) and $N$ cities (i.e., $N$ tasks). All salesmen start their journey from the depot and come back to the depot after visiting all cities (each city can be visited by only one salesman). The solution to mTSP is considered to be *complete* when all the cities have been visited, and all salesmen have returned to the depot. The min-max mTSP MDP is defined as:

**State.** We define state $s_\tau$ as the $\tau$-th partial solution of mTSP (i.e., the completed/uncompleted tasks, the status of agents, and the sequence of the past assignments). To be specific, $s_\tau = (\{s_\tau^i\}_{i=1}^{N+m}, s_\tau^{\text{env}})$ composed of two types of states: entity state $s_\tau^i$ and environment state $s_\tau^{\text{env}}$.

- $s_\tau^i = (p_\tau^i, 1_\tau^{\text{active}}, 1_\tau^{\text{assigned}})$ is the state of the $i$-th entity. $p_\tau^i$ is the 2D Cartesian coordinate of the $i$-th entity at the $\tau$-th event. $1_\tau^{\text{active}}$ indicates whether the $i$-th (i.e., agent/task is active agent is working/ task is not visited). Similarly, $1_\tau^{\text{assigned}}$ indicates whether the agent/task is assigned.
- $s_\tau^{\text{env}}$ contains the current time of the environment, and the sequences of cities visited by the salesmen.

The initial $s_0$ and terminal state $s_{\text{T}}$ are defined as an empty and a complete solution, respectively.

**Action.** We define action $a_\tau$ as the assignment of an idle agent to one of the feasible tasks (unassigned tasks). We refer to $a_\tau$ as the *agent-to-task assignment*. When multiple agents are idle at the same time $t$, we randomly choose one agent and assign a task to the agent. This is repeated until no agent is idle. Note that such randomness does not alter the resulting solutions, since the agents are considered to be homogeneous and the scheduling policy is shared.

**Transition**. The proposed MDP is formulated with an *event*-based transition. An event is defined as the case where any agent finishes the assigned task (e.g., a salesman reaches the assigned city in mTSP). Whenever an event occurs, the idle agent is assigned to a new task, and the status of the agent and the target task are updated accordingly. We enumerate the event with $\tau$ to avoid confusion from the elapsed time of the problem; $t(\tau)$ is a function that returns the time of event $\tau$.

**Reward**. The proposed MDP uses the negative makespan (i.e. total completion time of tasks) as the reward (i.e., $r(s_{\text{T}}) = -t(\text{T})$) that is realized only at $s_{\text{T}}$.

## 4 SCHEDULENET

In this section, we explain how ScheduleNet recommends $a_\tau$ of an idle agent from $s_\tau$. This is done by (1) constructing the agent-task graph $\mathcal{G}_\tau$, (2) embedding $\mathcal{G}_\tau$ using TGA, and (3) computing the assignment probabilities. Figure 1 illustrates the decision-making process of ScheduleNet.

### 4.1 Constructing Agent-task Graph

ScheduleNet constructs the *agent-task graph* $\mathcal{G}_\tau$ that reflects the complex relationships among the entities in $s_\tau$. Specifically, ScheduleNet constructs a directed complete graph $\mathcal{G}_\tau = (\mathbb{V}, \mathbb{E})$ out of $s_\tau$,

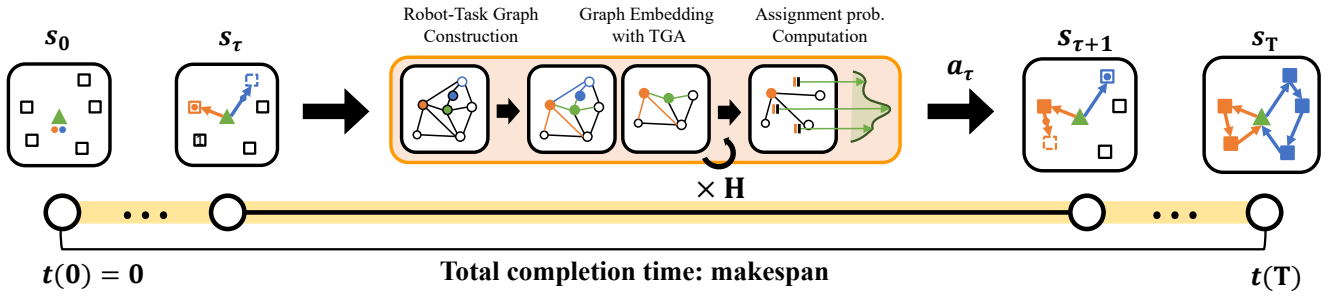



**Figure 1: Solving mTSP with ScheduleNet. At every event of the MDP, ScheduleNet constructs the agent-task graph $\mathcal{G}_\tau$ from $s_\tau$, then computes the node embedding of $\mathcal{G}_\tau$ using TGA, and finally computes the agent-task assignment probabilities from the node embedding. Green triangles and circles illustrate the depot. Blue and orange rectangles and circles illustrate agents. Arrows illustrate the tours of agents.**

where $\mathbb{V}$ is the set of nodes and $\mathbb{E}$ is the set of edges. The nodes and edges and their associated features are defined as:

- $v_i$ denotes the $i$-th node, which represents either an agent or a task. $v_i$ contains the node feature $x_i = (s_\tau^i, k_i)$, where $s_\tau^i$ is the state of entity $i$, and $k_i$ is the type of $v_i$. For example, if the entity $i$ is an agent and $1_\tau^{\text{active}} = 1$, then $k_i$ becomes *active-agent*. The list of all possible node types $k_i$ are (1) assigned-agent, (2) unassigned-agent (i.e., idle agent), (3) assigned-task, (4) unassigned-task, (5) inactive-task (i.e., visited city) and (6) depot.
- $e_{ij}$ denotes the edge between the source node $v_j$ and the destination node $v_i$. The edge feature $w_{ij} = (d_{ij}, k_{ij})$, where $d_{ij}$ is the Euclidean distance between $v_i$ and $v_j$, and $k_{ij} = [k_i, k_j]$ is the edge type.

In the following subsections, we omit the event iterator $\tau$ for notational brevity, since the action selection procedure is only associated with the current event index $\tau$.

## 4.2 Graph Embedding Using TGA

The agent-task graph $\mathcal{G}$ is a hetero graph with different node and edge types. The hetero graph is often processed via a Graph Neural Network (GNN), which updates nodes with type-specific node/edge updating functions [32, 37]. However, mTSP $\mathcal{G}$ has exceedingly many nodes and, especially, edge types. That necessitates us to maintain many independent node/edge updaters. Furthermore, the imbalanced distribution of node and edge types may hinder the representability and trainability of GNN. For example, $\mathcal{G}$ has one depot type while $N$ and $m$ number of city and salesmen nodes. That risks an unbalanced training of each update depending on the input graphs.

Furthermore, this imbalanced type distribution may also hinder message aggregation of GNN from extracting crucial information. For example, when the GNN aggregates the messages for the target agent node, there exist the messages from (at most) $N$ "city-agent" type edges and one message from the "city-depot" type edge. Hence, aggregating all messages from different typed edges can diminish the effect of the "city-depot" type edge after message aggregation. However, the "depot-agent" type edge is crucial in deciding whether the agent returns to the depot.

To alleviate the aforementioned issue, we propose Type-aware Graph Attention (TGA), which is designed to explicitly consider the type information and perform message aggregations for each edge type while sharing the parameters of per-type node/edge updaters.

ScheduleNet computes the node embeddings from the agent-task graph $\mathcal{G}$ using TGA, designed to capture the different relations among the graph entities by applying attention mechanisms for each relational type. TGA uses three steps to compute the updated node/edge embedding as follows:

**Type-aware edge update.** The edge update scheme is designed to reflect the complex type relationship among the entities while updating edge features. First, the context embedding $c_{ij}$ of edge $e_{ij}$ is computed using the source and destination node type $k_i, k_j$ such that:

$$c_{ij} = [\text{Emb}(k_i), \text{Emb}(k_j)] \tag{1}$$

where $\text{Emb}(\cdot)$ is a trainable lookup table function (i.e., each type $k_i$ is modeled via a trainable vector). Next, the type-aware edge encoding $u_{ij}$ is computed using a multilayer perceptron (MLP) as follows:

$$u_{ij} = \text{MLP}_{etype}(h_i, h_j, h_{ij}, c_{ij}) \tag{2}$$

where $\text{MLP}_{etype}(\cdot)$ is the type-aware edge encoding MLP. $u_{ij}$ can be seen as a dynamic edge feature that varies depending on the source and destination node type. Then, the updated edge embedding $h'_{ij}$ and its attention logit $z_{ij}$ are obtained as:

$$h'_{ij} = \text{MLP}_{edge}(u_{ij}) \tag{3}$$

$$z_{ij} = \text{MLP}_{attn}(u_{ij}) \tag{4}$$

where $\text{MLP}_{edge}$ and $\text{MLP}_{attn}$ is the edge updater and logit function, respectively. The edge updater and logit function produce updated edge embedding and logits from the type-aware edge.

**Type-aware message aggregation.** Each entity in the agent-task graph interacts differently with the other entities, depending on the type of edges between them. To preserve the different relationships among the entities during the graph embedding procedure, TGA gathers messages $h'_{ij}$ via the type-aware message aggregation.

First, TGA aggregates messages for each node type and produces the *per-type* message $m_i^k$ as follows:

$$m_i^k = \sum_{j \in \mathcal{N}_k(i)} \alpha_{ij} h'_{ij} \tag{5}$$

where $\mathcal{N}_k(i) = \{v_l | k_l = k, v_l \in \mathcal{N}(i)\}$ is the type $k$ neighborhood of $v_i$, and $\alpha_{ij}$ is the attention score that is computed using $z_{ij}$:

$$\alpha_{ij} = \frac{\exp(z_{ij})}{\sum_{j \in \mathcal{N}_k(i)} \exp(z_{ij})} \tag{6}$$

Intuitively speaking, The proposed attention scheme normalizes the attention logits of incoming edges over the types. Therefore, the attention scores sum up to 1 over each type $k$ neighborhood. TGA aggregates the per-type messages to compute the total aggregated message $m_i$ for $v_i$ as:

$$m_i = \sum_{k \in \mathbb{K}} m_i^k \tag{7}$$

where $\mathbb{K}$ is the set of node types.

**Type-aware node update.** Similar to the edge update phase, the context embedding $c_i$ is computed first for each node $v_i$:

$$c_i = \text{Emb}(k_i) \tag{8}$$

where $\text{Emb}(\cdot)$ is a trainable lookup table function. Then, the updated hidden node embedding $h'_i$ is computed as below:

$$h'_i = \text{MLP}_{node}(h_i, m_i, c_i) \tag{9}$$

where $\text{MLP}_{node}(\cdot)$ is the type-aware node updater.

ScheduleNet computes the node embeddings from $\mathcal{G}$ using TGA. The embedding procedure first encodes the features of $\mathcal{G}$ into the initial node embeddings $\{h_i^{(0)} | v_i \in \mathbb{V}\}$, and the initial edge embeddings $\{h_{ij}^{(0)} | e_{ij} \in \mathbb{E}\}$ by using linear projections. ScheduleNet then performs TGA $H$ times on all nodes to compute the final node embeddings $\{h_i^{(H)} | v_i \in \mathbb{V}\}$ and edge embeddings $\{h_{ij}^{(H)} | e_{ij} \in \mathbb{E}\}$ by using the shared one TGA layer.

## 4.3 Computing Assignment Probability

Using $\{h_i^{(H)} | v_i \in \mathbb{V}\}$ and $\{h_{ij}^{(H)} | e_{ij} \in \mathbb{E}\}$, ScheduleNet selects the assignment action $a_\tau$ for the target idle agent. It computes the assignment probability of the target agent $i$ to the *unassigned* task $j$ as follows:

$$l_{ij} = \text{MLP}_{actor}(h_i^{(H)}, h_j^{(H)}, h_{ij}^{(H)}) \tag{10}$$

$$p_{ij} = \text{softmax}(\{l_{ij}\}_{j \in \mathbb{A}(\mathcal{G}_\tau)}) \tag{11}$$

where $\mathbb{A}(\mathcal{G}_\tau)$ denotes a set of feasible actions that is defined as $\{v_j | k_j = \textit{Unassigned-task}, v_j \in \mathbb{V}\}$.

The proposed MDP formulated mTSP is designed to have only one target idle agent at a given event $\tau$. Due to the design of MDP, computing one agent's action probability becomes enough for deciding an action at each $\tau$. This allows us to solve mTSP without explicitly modeling the joint actions of agents, unlike many multi-agent RL approaches.

# 5 TRAINING SCHEDULENET

We utilize the sparse team reward as the reward to train the scalable scheduler that aims to complete the tasks as quickly as possible by coordinating multiple agents. Even though this team reward is the most direct signal that can be used for solving min-max mTSP, training a cooperative policy using a single sparse and delayed reward is notoriously difficult [13, 31]. The high variance of the reward, due to the combinatorial nature of mTSP, adds an additional difficulty. To handle such difficulties, we employ two training stabilizers, reward normalization and Clip-REINFORCE.

## 5.1 Reward Normalization

We denote the makespan induced by policy $\pi_\theta$ as $M(\pi_\theta)$. We observe that $M(\pi_\theta)$ is highly volatile depending on the problem size $(N, m)$ and $\pi_\theta$. To reduce the variance of the reward incurred from the problem size, we propose to use the normalized makespan $\bar{M}(\pi_\theta, \pi_b)$ computed as:

$$\bar{M}(\pi_\theta, \pi_b) = \frac{M(\pi_\theta) - M(\pi_b)}{M(\pi_b)} \tag{12}$$

where $\pi_\theta$ and $\pi_b$ are the current and baseline policies, respectively.

A similar normalization scheme that only measures the performance difference between $\pi$ and $\pi_b$ has been investigated in RL applications for solving single-agent scheduling problems [6, 20, 21]. Such normalization can provide consistent learning signals when the training instance sizes (i.e. $N$) are fixed, which is a common practice for training RL methods to solve single-agent scheduling problems. However, we observed that even if $N$ is fixed, the (optimal) makespan of mTSP can differ severely as $m$ changes. To reduce the variability of the makespan due to $m$, we further divide the makespan difference by $M(\pi_b)$.

**Effect of the baseline selection.** A proper selection of $\pi_b$ is essential to assure stable and asymptotically better learning of ScheduleNet. Intuitively speaking, choosing too strong baseline (i.e., policy having smaller makespan such as LKH3 and OR-tools) can make the entire learning process unstable since the normalized reward tends to have larger values. On the other hand, employing too weak baseline can lead to virtually no learning since the $\bar{M}(\pi, \pi_b)$ becomes nearly zero.

We select $\pi_b$ as $\text{Greedy}(\pi_\phi)$, where $\pi_\phi$ is the soft target network of $\pi_\theta$ as proposed in [23] and $\text{Greedy}(\pi)$ is the argmax of $\pi$. This baseline selection has several advantages from selecting a fixed/pre-existing scheduling policy: (1) Entire learning process becomes independent from existing scheduling methods and trainable via solely RL. (2) $\text{Greedy}(\pi_\phi)$ serves as an appropriate $\pi_b$ (either not too strong or weak) during policy learning. We experimentally confirmed that the baseline section $\text{Greedy}(\pi_\phi)$ results in a better scheduling policy similar to the several literature [20, 34].

Using $\bar{M}(\pi_\theta, \pi_b)$, we compute the normalized return $G_\tau(\pi_\theta, \pi_b)$ as follows:

$$G_\tau(\pi_\theta, \pi_b) = -\gamma^{T-\tau} \bar{M}(\pi_\theta, \pi_b) \tag{13}$$

where $T$ is the index of the terminal state, and $\gamma$ is the discount factor of MDP. The minus sign is for minimizing the makespan. Note that, in the early phase of mTSP (when $\tau$ is small), it is difficult to estimate the makespan. Thus, we place a smaller weight (i.e, $\gamma^{T-\tau}$) on $\bar{M}(\pi_\theta, \pi_b)$, which is evaluated when $\tau$ is small (early stage).

### 5.2 Clip-REINFORCE

Even a small change in a single assignment can result in a dramatic change to the makespan due to the combinatorial nature of mTSP. Hence, training the value function that predicts $G_\tau$ reliably is difficult. We thus propose to utilize Clip-REINFORCE, a variant of PPO [33] *without* the learned value function, for training ScheduleNet. The objective of the Clip-REINFORCE is given as follows:

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{(\mathcal{G}_\tau, a_\tau)\sim\pi_\theta} [\min(\text{clip}(\rho_\tau, 1-\epsilon, 1+\epsilon)G_\tau, \rho_\tau G_\tau)] \quad (14)$$

where $\text{clip}(x, a, b) = \{a \text{ if } x \le a, x \text{ if } a < x < b, b \text{ if } x \ge b\}$, $\epsilon$ is the clipping parameter, $G_\tau$ is a shorthand notation for $G_\tau(\pi_\theta, \pi_b)$, and $\rho_\tau = \pi_\theta(a_\tau|\mathcal{G}_\tau)/\pi(a_\tau|\mathcal{G}_\tau)$.

## 6 EXPERIMENTS

In this section, we evaluate the performance of ScheduleNet on mTSP. To calculate the inference time, we run all experiments on the server equipped with AMD Threadripper 2990WX CPU. We use a single CPU core for evaluating all algorithms.

### 6.1 mTSP Experiments

We denote $(N \times m)$ as the mTSP with $N$ cities (tasks) and $m$ salesmen (agents). To generate a random mTSP instance, we sample $N$ and $m$ from $U(15, 30)$ and $U(3, 4)$, respectively. Similarly, the Euclidean coordinates of $N$ cities are sampled uniformly from the unit square. ScheduleNet is trained on random mTSP instances that are generated on-the-fly. For all experimental results, we evaluate the performance of the trained ScheduleNet model without any additional training (i.e., zero-shot performances).

**Training details.** ScheduleNet is composed of three components. The first components generates the initial node embedding $\{h_i^{(0)}\}$ and edge $\{h_{ij}^{(0)}\}$ by using linear projections. The second component (TGA), which utilizes MLP(64) as $\text{MLP}_{\text{etype}}$, $\text{MLP}_{\text{edge}}$, $\text{MLP}_{\text{attn}}$, and $\text{MLP}_{\text{node}}$, repeatedly updates the node and edge embedding 3 times to generate $\{h_i^{(3)}\}$ and edge $\{h_{ij}^{(3)}\}$. The last component $\text{MLP}_{\text{actor}}$, which is parametrized as MLP(64, 32), generates action logits from the set of node and edge embedding. All input and output dimensions of MLPs are 64 and hidden actions are LeakyReLU. We then present a pseudocode for training ScheduleNet (see Algorithm 2). We set the MDP discounting factor $\gamma$ to 0.99, learning rate $\alpha$ to 0.0001, $E$ to 128, $K$ to 4, $\beta$ to 0.01, and the clipping parameter $\epsilon$ of Clip-REINFORCE to 0.2. We set the training seed as 1234.

**Results on random mTSP datasets.** To investigate the generalization capacity of ScheduleNet to various problem sizes, we evaluate ScheduleNet on the randomly generated mTSP datasets. Each $(N \times m)$ dataset consists of 100 random uniform mTSP instances.

We consider four baseline algorithms to measure the performance of ScheduleNet. As the non-learning baselines, LKH3 [14] and Google OR-Tools [29] are used. It is noteworthy that LKH3 is known to find the optimal solutions for mTSP problems which have known optimal solutions. Thus, we use the makespans computed by LKH3 as the proxies for the optimal solutions. As the RL baselines, GNN-DisPN [15] and DAN [6] are considered. To the best of our knowledge, these are the only RL algorithms that tried to solve the min-max mTSP.

**Algorithm 2:** ScheduleNet Training

**input :** Training policy $\pi_\theta$, baseline policy $\pi_\phi$
**output:** Optimized policy $\pi_\theta$

Initialize the baseline policy with parameters $\phi \leftarrow \theta$
**for** *update step* **do**
  Initialize sample buffer $\mathcal{D} \leftarrow \emptyset$
  **for** *number of episodes E* **do**
    Generate a random mTSP instance $I$
    $\pi_b \leftarrow \text{Greedy}(\pi_\phi)$
    Construct mTSP MDP from the instance $I$
    Collect samples $\mathcal{S} = \{\mathcal{G}_\tau, a_\tau, \pi_{\theta_{\text{old}}}(a_\tau|\mathcal{G}_\tau), G_\tau(\pi_\theta, \pi_b)\}_{\tau=0}^{T}$ with $\pi_\theta$ and $\pi_b$ from the MDP.
    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{S}$.
  **end**
  **for** *inner updates K* **do**
    Calculate the loss $\mathcal{L}(\theta)$ with $\mathcal{D}$
    $\theta \leftarrow \theta + \alpha\nabla_\theta\mathcal{L}(\theta)$
  **end**
  $\phi \leftarrow \beta\phi + (1-\beta)\theta$
**end**
**return** $\pi_\theta$

Table 1 shows the average makespans and average gaps (i.e., the relative makespan w.r.t LKH3) of ScheduleNet and the baseline algorithms on various-sized random mTSP instances. ScheduleNet generally shows leading performances compared to the baseline algorithms even though it is trained on the smallest test setup (30×3). In addition, Figure 2 compares the average computation times and their standard deviations for each problem size, clearly indicating that ScheduleNet is significantly faster than traditional heuristic solvers (LKH3 and OR-Tools). The computational time difference among RL approaches is almost not noticeable. We also provide the experiment results for larger datasets in Table 2.

**Results on public benchmarks.** Next, to explore the generalization of ScheduleNet to problems that come from completely different distributions (e.g., real-world data), we present the results on the mTSPLib dataset defined by Necula et al. [28]. Note that mTSPLib consists of four instances of size 51, 52, 76 and 99 from TSPLib [30], each of which is extended to multi-agent setups where $m$ equals to 2, 3, 5, and 7.

Table 3 shows the makespans and average gaps w.r.t CPLEX of the algorithms. ScheduleNet still exhibits comparable performance to OR-tools, while DAN shows significant performance drops. We further inspect the solutions of DAN and ScheduleNet in *eli51-2* and *belin52-2* where the cities are distributed (relatively) uniformly and non-uniformly, respectively. As shown in Figure 3a, both learning algorithm shows similar solution patterns in *eli51-2*. However, for non-uniformly distributed cities (*belin52-2*), the solution quality of DAN deteriorates (see Figure 3b). ScheduleNet performs better than DAN, even when the city distributions differ from the training. From the results, we can observe that ScheduleNet is effective in

**Table 1: Random mTSP results. GNN-DisPN results are reproduced from Hu et al. [15]. (g.) indicate the results of greedy decoding, (s.n) indicates the result of best costs of $n$ sampled.**

| $N$ | 30 | | 50 | | | 100 | | | 200 | | | Gap |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $m$ | 3 | 5 | 5 | 7 | 10 | 5 | 10 | 15 | 10 | 15 | 20 | |
| LKH3 | 2.17 | 1.94 | 2.00 | 1.95 | 1.91 | 2.20 | 1.97 | 1.98 | 2.04 | 2.00 | 1.97 | 1.00 |
| OR-Tools | 2.25 | 1.95 | 2.04 | 1.96 | 1.96 | 2.36 | 2.29 | 2.25 | 2.57 | 2.59 | 2.59 | 1.12 |
| GNN-DisPN | - | - | 2.12 | - | 1.95 | 2.48 | 2.09 | - | - | - | - | - |
| DAN (g.) | 2.58 | 2.11 | 2.29 | 2.11 | 2.03 | 2.72 | 2.17 | 2.09 | 2.40 | 2.20 | 2.15 | 1.12 |
| DAN (s.64) | 2.32 | 2.00 | 2.12 | 1.99 | 1.95 | 2.55 | 2.05 | 2.00 | 2.29 | 2.13 | 2.07 | 1.05 |
| ScheduleNet (g.) | 2.32 | 2.02 | 2.17 | 2.07 | 1.98 | 2.59 | 2.13 | 2.07 | 2.45 | 2.24 | 2.17 | 1.09 |
| ScheduleNet (s.64) | 2.22 | 1.96 | 2.07 | 1.99 | 1.92 | 2.43 | 2.03 | 1.99 | 2.25 | 2.08 | 2.05 | 1.04 |

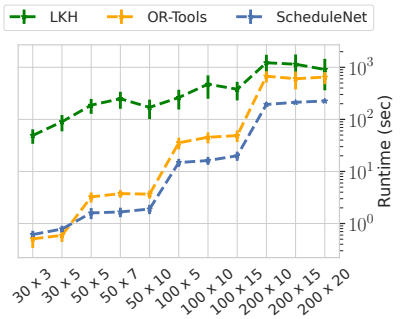
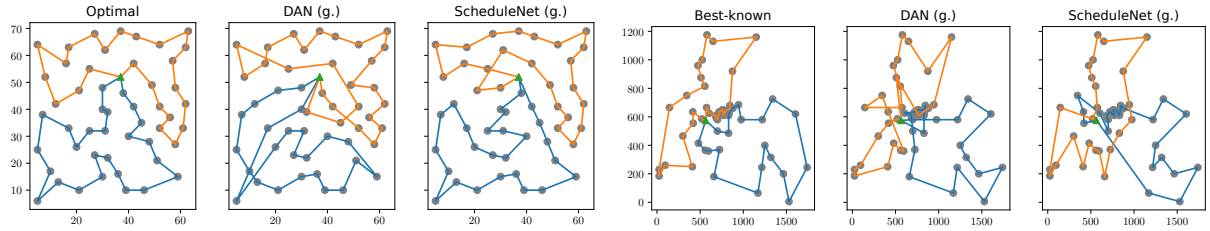



**Figure 2: mTSP runtimes. X-axis shows the size of mTSP instances and Y-axis shows the run time (in seconds) of each algorithm.**




**(a) Eli51-2 results** **(b) Berlin52-2 results**

**Figure 3: Example routes of Optimal (or Best-known) solutions, DAN, and ScheduleNet (a) Greedy decoding results of DAN and ScheduleNet on Eli52-2. (b) Greedy decoding results of DAN and ScheduleNet on Berlin52-2.**

**Table 2: Extended random mTSP results. (g.) indicate the results of greedy decoding.**

| $N \times m$ | $300 \times 30$ | $500 \times 50$ | $700 \times 50$ | Gap |
|---|---|---|---|---|
| LKH | 2.05 | 2.94 | 2.25 | 1.00 |
| OR-Tools | 2.94 | 7.55 | 10.90 | 3.45 |
| ScheduleNet (g.) | 2.25 | 2.31 | 2.50 | 1.14 |

solving both randomly generated and real-world mTSPs problems (i.e., robust transferability over data distribution shift).

## 6.2 mTSP Variants Experiments

While most state-of-the-art neural (multi-agent) schedulers focus on solving classical scheduling problems, they overlook the application of such methods in practical scenarios. In this section, we further investigate the performance of ScheduleNet in more practical scenarios of mTSP: (1) limited observation and (2) online routing.

**Limited observation.** In a real-world application of mTSP, some salesmen may not be able to gather all the information about the other salesmen due to the limited communication capabilities (e.g., delivery trucks are located in distant). In such a scenario, the agent should decide the next city to visit with local observations. To consider this realistic scenario, we limit the number of observable salesmen $N_r$ from $\mathcal{G}_\tau$ and investigate performances.

We employ ScheduleNet to solve the test instances, whose sizes are (50×5), (100×10), and (200×20), by varying $N_r$. As shown in Figure 4, the makespan decreases as $N_r$ increases because the extended communication scope (i.e., close to the global observation) can induce enhanced coordination among salesmen. Note that $N_r = 0$ indicates the case where each salesman makes entirely independent actions without considering the other salesmen at all. It is noteworthy that ScheduleNet maintains the scheduling performance when a few numbers of salesmen are observable. The results imply that ScheduleNet learns an effective policy that is operated on partial observations and performs robustly even with limited communication capabilities (i.e., the lack of global observation).

**Online routing.** Unlike the neural methods that are trained for generating complete scheduling solutions [6, 20], ScheduleNet can solve the dynamic scheduling problem, where agents and cities appear during the execution of scheduled actions, in an online manner, due to our MDP formulation. We evaluate this capability by solving the online mTSP problem where new cities appear dynamically within the scheduling problems.

To generate the online scenarios, we first fix the number of cities added during the online routing. Once the number of cities is given, we then decide the locations of the cities by taking the midpoints of cities. We assign the location of the city that will be first added as the midpoint between the first and second cities of the mTSP map. Similarly, the location of the second city is set as the

**Table 3: mTSPLib results. CPLEX results with $*$ are optimal solutions. Otherwise, the best-known upper bound of CPLEX results are reported.**

| Instance | *eil51* | | | | *berlin52* | | | | *eil76* | | | | *rat99* | | | | Gap |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $m$ | 2 | 3 | 5 | 7 | 2 | 3 | 5 | 7 | 2 | 3 | 5 | 7 | 2 | 3 | 5 | 7 | |
| CPLEX | 222.7* | 159.6 | 124.0 | 112.1 | 4110.2 | 3244.4 | 2441.4 | 2440.9 | 280.9* | 197.3 | 150.3 | 139.6 | 728.8 | 587.2 | 469.3 | 443.9 | 1.00 |
| LKH3 | 222.7 | 159.6 | 124.0 | 112.1 | 4110.2 | 3244.4 | 2441.4 | 2440.9 | 280.9 | 197.3 | 150.3 | 139.6 | 728.8 | 587.2 | 469.3 | 443.9 | 1.00 |
| OR-Tools | 243.3 | 170.5 | 127.5 | 112.1 | 4665.5 | 3311.3 | 2482.6 | 2440.9 | 318.0 | 212.4 | 143.4 | 128.3 | 762.2 | 552.1 | 473.7 | 442.5 | 1.03 |
| DAN (g) | 274.2 | 178.9 | 158.6 | 118.1 | 5,226.0 | 4,278.4 | 2,758.8 | 2,696.8 | 361.1 | 251.5 | 170.9 | 148.5 | 930.8 | 674.1 | 504.0 | 466.4 | 1.18 |
| DAN (s. 64) | 252.9 | 178.9 | 128.2 | 114.3 | 5,097.7 | 3,455.7 | 2,677.1 | 2,494.5 | 336.7 | 228.1 | 157.9 | 134.5 | 966.5 | 697.7 | 495.6 | 462.0 | 1.11 |
| ScheduleNet (g.) | 257.9 | 185.3 | 131.7 | 113.9 | 4,826.1 | 3,644.2 | 2,747.6 | 2,514.6 | 330.2 | 228.8 | 163.9 | 144.4 | 843.8 | 650.8 | 500.7 | 469.9 | 1.13 |
| ScheduleNet (s.64) | 239.3 | 173.5 | 125.8 | 112.2 | 4,591.6 | 3,276.1 | 2,517.3 | 2,441.4 | 317.7 | 220.8 | 153.8 | 131.7 | 781.2 | 627.1 | 502.3 | 464.4 | 1.05 |

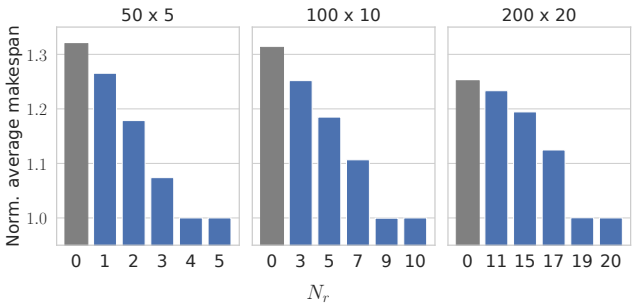



**Figure 4: Limited observation scenario results. X-axis shows $N_r$ (i.e., the number of observable agents), and Y-axis shows the normalized average makespans.**

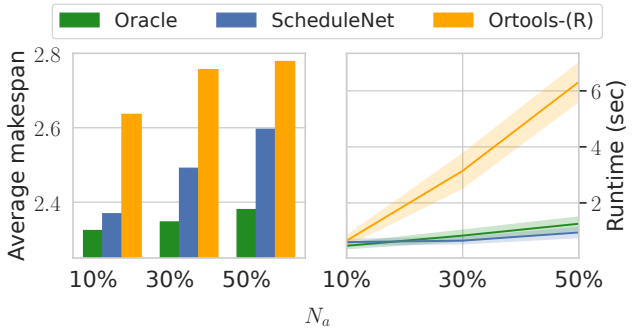
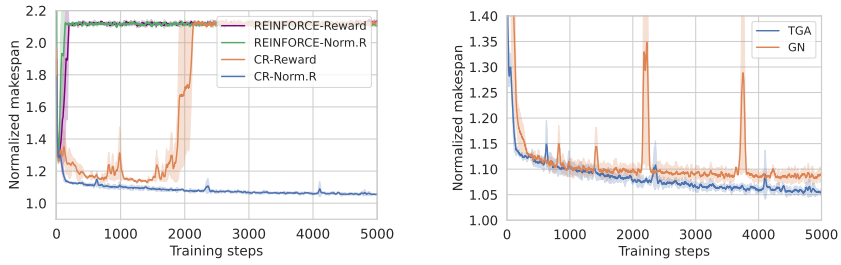



**Figure 5: Online routing scenario results. [Left] Average makespans of algorithms in online routing. X-axis shows $N_a$ (i.e., percentage of added cities), and Y-axis shows average makespans. [Right] Increased run times over $N_a$. X-axis shows $N_a$, and Y-axis shows average run times in seconds. The solid lines visualize the average runtimes, and shadow areas visualize the standard deviation of the runtimes.**




**(a) Training method ablation** **(b) Representation method ablation**

**Figure 6: Normalized makespan curves over training steps. (a) Training method ablation results. (b) Representation method ablation results. X-axis shows the training steps. Y-axis shows the averaged normalized makespans of validation instances during training. The shadow regions visualize the standard deviations of normalized makespans over five different random seeds.**

**Figure 7: Representation method ablation (makespans on the large instances)**

| $N$ | $m$ | GN | TGA |
|---|---|---|---|
| 100 | 5 | 2.67 | **2.59** |
| | 10 | 2.26 | **2.13** |
| | 15 | 2.36 | **2.07** |
| 200 | 10 | 2.61 | **2.45** |
| | 15 | 2.41 | **2.24** |
| | 20 | 2.52 | **2.17** |

midpoint between the second and third cities. Using the same logic, we decide all positions of the newly added cities. Lastly, we decide the timing of city addition by using (near) optimal solutions (e.g., LKH3) $M^*(\pi)$. We first compute divide $M^*(\pi)$ by $1.0 < s$ and then evenly allocate the timing of additions from 0.0 to $M^*(\pi)/s$. This ensures all "to be added" cities are added during the online routing simulation, no matter what scheduling policies are evaluated. We set $s = 2$.

We first randomly generate 100 (30×3) test instances. Then, while simulating these scenarios with the scheduling policies, we sequentially add $30 \times N_a\%$ cities to the simulation (the location and times of the cities are unknown to policies before the addition) and evaluate how the policies effectively replan the schedules according to such online scenarios. We compare the makespan of ScheduleNet with (1) oracle planning that plans once while knowing all future demands (its appearing locations and times), and (2) re-planing heuristics that

replans the future actions whenever $s_\tau$ is updated. To implement the oracle planning, we reformulate the online routing problem as a vehicle routing problem with time constraints (VRP-TW). In the VRP-TW, we set the time window constraints for the originally existing cities as (0, $M$), where $M$ is a large number. For the newly added cities, the lower and upper bounds of the time window are set as their addition times and $M$. The re-planing heuristics are implemented with OR-tools [29].

Figure 5 shows the average makespans computed using 100 random online scenarios. As shown in the left plot, ScheduleNet always produces a shorter makespan and shorter runtime than the replanning heuristic for all $N_a$ values, proving its effective adaptability and robustness to dynamically changing cities. Note that as $N_a$ increases, the makespan and the runtime increase for all methods due to the increased uncertainty and the number of replanning.

## 7 ABLATION STUDIES

In this paper, we propose TGA as the representation learning module and Reward normalization and Clip-REINFORCE (CR) as the training methods. Here, we provide the experimental results that show the effect of the proposed representation module and training methods on the training and testing performance of ScheduleNet. We first validate the effectiveness of the proposed training methods by evaluating the following ScheduleNet variants:

- `REINFORCE-Reward`: the model receives unnormalized reward and is trained by REINFORCE [38].
- `REINFORCE-Norm.R`: the model receives normalized reward and is trained by REINFORCE
- `CR-Reward`: the model receives normalized reward and is trained by CR
- `CR-Norm.R (ScheduleNet)`: the model receives normalized reward and is trained by CR

Figure 6a compares the average normalized makespans (over the five independent runs) of the models on the test instances (size of 30×3) during training. As shown in Figure 6a, only `CR-Norm.R (ScheduleNet)` successfully converges to the lowest makespans. On the other hand, the other models minimize makespans at the early training phase; however, their performances start to deteriorate and converge to ∼2.1 as the training proceeds. From the experimental results, we can confirm that the proposed training methods stabilize the training of ScheduleNet.

We then validate the effectiveness of the proposed representation module (i.e., TGA) compared to GN, a well-known node, and edge encoding GNN layer [3]. We evaluate the Schedulent variants of the following:

- `GN`: the model using the graph network (GN) layer is trained by CR with the normalized reward.
- `TGA` (ScheduleNet): the model using TGA and is trained by CR with the normalized reward.

By comparing the learning curves in Figure 6b, we can see the effect of TGA as a representation module. TGA induces more stabilized and faster performance improvement. In addition, as shown in Table 7, `TGA` (ScheduleNet) consistently shows better performance than `GN` in the test datasets. The performance difference between the two models becomes severe as $N$ and $m$ increase.

## 8 CONCLUSION

In this work, we proposed ScheduleNet, an RL-based scheduler that solves the min-max mTSP in a scalable manner. The core components of ScheduleNet are (1) the agent-task state representation that is effective in solving mTSP, (2) the type-aware graph attention (TGA) network to effectively embed the agent-task graph, and (3) Clip-REINFORCE that stably learns the cooperative and scalable policy using sparse team reward (makespan). Through extensive experiments, we empirically verified that ScheduleNet effectively solves the classical static min-max mTSP while showing exceptional generalization to the change of the number of cities and city coordinate distribution. Furthermore, ScheduleNet also showed that it solves the limited communication or online mTSP variants.

## ACKNOWLEDGMENTS

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (2022-0-01032, Development of Collective Collaboration Intelligence Framework for Internet of Autonomous Things)

## REFERENCES

[1] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. 2020. Learning What to Defer for Maximum Independent Sets. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 134–144.
[2] Sungjun Ann, Youdan Kim, and Jaemyung Ahn. 2015. Area allocation algorithm for multiple UAVs area coverage based on clustering and graph method. *IFAC-PapersOnLine* 48, 9 (2015), 204–209.
[3] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks.
[4] Tolga Bektas. 2006. The multiple traveling salesman problem: an overview of formulations and solution procedures. *omega* 34, 3 (2006), 209–219.
[5] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural Combinatorial Optimization with Reinforcement Learning. https://openreview.net/forum?id=rJY3vK9eg
[6] Yuhong Cao, Zhanhong Sun, and Guillaume Sartoretti. 2021. DAN: Decentralized Attention-based Neural Network to Solve the MinMax Multiple Traveling Salesman Problem. *arXiv preprint arXiv:2109.04205* (2021).
[7] Omar Cheikhrouhou and Ines Khoufi. 2021. A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy. *Computer Science Review* 40 (2021), 100369.
[8] Xinyun Chen and Yuandong Tian. 2019. Learning to Perform Local Rewriting for Combinatorial Optimization. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc.
[9] Paulo R d O da Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. 2020. Learning 2-opt Heuristics for the Traveling Salesman Problem via Deep Reinforcement Learning. In *Proceedings of The 12th Asian Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 129)*, Sinno Jialin Pan and Masashi Sugiyama (Eds.). PMLR, Bangkok, Thailand, 465–480.
[10] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. 2021. Generalize a Small Pretrained Model to Arbitrarily Large TSP Instances. arXiv:2012.10658 [cs.LG]
[11] Kenneth C Gilbert and Ruth B Hofstra. 1992. A new multiperiod multiple traveling salesman problem with heuristic and application to a scheduling problem. *Decision Sciences* 23, 1 (1992), 250–259.
[12] Samuel Gorenstein. 1970. Printing press scheduling for multi-edition periodicals. *Management Science* 16, 6 (1970), B–373.
[13] Joshua Hare. 2019. Dealing with sparse rewards in reinforcement learning. *arXiv preprint arXiv:1910.09281* (2019).
[14] Keld Helsgaun. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University* (2017).
[15] Yujiao Hu, Yuan Yao, and Wee Sun Lee. 2020. A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs. *Knowledge-Based Systems* 204 (2020), 106244.

[16] Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, Thomas Laurent, and Xavier Bresson. 2020. Learning TSP Requires Rethinking Generalization. arXiv:2006.07054 [cs.LG]
[17] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*. 6348–6358.
[18] Minsu Kim, Jinkyoo Park, et al. 2021. Learning Collaborative Policies to Solve NP-hard Routing Problems. *Advances in Neural Information Processing Systems* 34 (2021).
[19] Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. 2021. Deep Policy Dynamic Programming for Vehicle Routing Problems. arXiv:2102.11756 [cs.LG]
[20] Wouter Kool, Herke van Hoof, and Max Welling. 2019. Attention, Learn to Solve Routing Problems!. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ByxBFsRqYm
[21] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. 2020. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. *Advances in Neural Information Processing Systems* 33 (2020).
[22] Jan Karel Lenstra and AHG Rinnooy Kan. 1975. Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society* 26, 4 (1975), 717–733.
[23] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning.. In *ICLR (Poster)*. http://arxiv.org/abs/1509.02971
[24] Hao Lu, Xingwen Zhang, and Shuang Yang. 2020. A Learning-based Iterative Method for Solving Vehicle Routing Problems. In *International Conference on Learning Representations*.
[25] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. 2020. Reinforcement learning for combinatorial optimization: A survey.
[26] LIU Ming and ZHANG Pei-yong. 2014. New hybrid genetic algorithm for solving the multiple traveling salesman problem: an example of distribution of emergence materials. *Journal of Systems & Management* 23, 2 (2014), 247.
[27] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. 2018. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*. 9839–9849.
[28] Raluca Necula, Mihaela Breaban, and Madalina Raschip. 2015. Tackling the bi-criteria facet of multiple traveling salesman problem with ant colony systems. In *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 873–880.
[29] Laurent Perron and Vincent Furnon. 2022. *OR-Tools*. Google. https://developers.google.com/optimization/
[30] Gerhard Reinelt. 1991. TSPLIB—A traveling salesman problem library. *ORSA journal on computing* 3, 4 (1991), 376–384.
[31] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. 2018. Learning by playing solving sparse reward tasks from scratch. In *International Conference on Machine Learning*. PMLR, 4344–4353.
[32] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*. Springer, 593–607.
[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms.
[34] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.
[35] Joseph A Svestka and Vaughn E Huckfeldt. 1973. Computational experience with an m-salesman traveling salesman algorithm. *Management Science* 19, 7 (1973), 790–799.
[36] Lixin Tang, Jiyin Liu, Aiying Rong, and Zihou Yang. 2000. A multiple traveling salesman problem model for hot rolling scheduling in Shanghai Baoshan Iron & Steel Complex. *European Journal of Operational Research* 124, 2 (2000), 267–282.
[37] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The world wide web conference*. 2022–2032.
[38] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.
[39] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. 2020. Learning Improvement Heuristics for Solving Routing Problems. arXiv:1912.05784 [cs.AI]