electronics

MDPI

Article

Scalable Multi-Robot Task Allocation Using Graph Deep Reinforcement Learning with Graph Normalization

Zhenqiang Zhang [1], Xiangyuan Jiang [1], Zhenfa Yang [2], Sile Ma [1,2,*], Jiyang Chen [1,3] and Wenxu Sun [1,*]

[1] Institute of Marine Science and Technology, Shandong University, Qingdao 266237, China; 202020867@mail.sdu.edu.cn (Z.Z.); xyjiang@sdu.edu.cn (X.J.); chenjiyang@sdu.edu.cn (J.C.)
[2] School of Control Science and Engineering, Shandong University, Jinan 250061, China; yangzhenfa@sdu.edu.cn
[3] Shandong Zhengzhong Information Technology Co., Ltd., Jinan 250098, China
* Correspondence: masile@sdu.edu.cn (S.M.); sunwenxu@sdu.edu.cn (W.S.)

Abstract: Task allocation plays an important role in multi-robot systems regarding team efficiency. Conventional heuristic or meta-heuristic methods face difficulties in generating satisfactory solutions in a reasonable computational time, particularly for large-scale multi-robot task allocation problems. This paper proposes a novel graph deep-reinforcement-learning-based approach, which solves the problem through learning. The framework leverages the graph sample and aggregate concept as the encoder to extract the node features in the context of the graph, followed by a cross-attention decoder to output the probability that each task is allocated to each robot. A graph normalization technique is also proposed prior to the input, enabling an easy adaption to real-world applications, and a deterministic solution can be guaranteed. The most important advantage of this architecture is the scalability and quick feed-forward character; regardless of whether cases have a varying number of robots or tasks, single depots, multiple depots, or even mixed single and multiple depots, solutions can be output with little computational effort. The high efficiency and robustness of the proposed method are confirmed by extensive experiments in this paper, and various multi-robot task allocation scenarios demonstrate its advantage.

Keywords: multi-robot task allocation; graph deep reinforcement learning; graph sample and aggregate; cross-attention; graph normalization; single depot; multiple depot

Citation: Zhang, Z.; Jiang, X.; Yang, Z.; Ma, S.; Chen, J.; Sun, W. Scalable Multi-Robot Task Allocation Using Graph Deep Reinforcement Learning with Graph Normalization. *Electronics* **2024**, *13*, 1561. https://doi.org/10.3390/electronics13081561

Academic Editor: Domenico Rosaci

Received: 3 March 2024
Revised: 11 April 2024
Accepted: 17 April 2024
Published: 19 April 2024

1. Introduction

Automatic mobile robotics in large-scale mission contexts is encouraging the use of multi-robot systems (MRSs) due to their superior efficiency, scalability, and robustness compared to single-robot systems [1]. One of the most important and challenging problems of MRSs is multi-robot task allocation (MRTA), which involves assigning a set of robots to a set of tasks while being subject to some constraints, with the objective of optimizing the overall system performance like the make-span, total energy consumption, etc. [2]. MRTA can be categorized into three areas [3]: (1) single-task robot (ST) versus multi-task robot (MT) allocation, from the aspect of the robot's capacity to execute a number of tasks at a time; (2) single-robot task (SR) versus multi-robot task (MR) allocation, from the aspect of the how many robots are required to finish a task; (3) instantaneous assignment (IA) versus time-extended assignment (TA) allocation, from the aspect of the assignment timing, where IA only instantaneously allocates tasks to robots without planning for future allocations, while TA produces the overall information about the assignment, including the sub-tasks set and the execution sequence of the tasks that each robot has been allocated to. Of the categories, ST-SR-IA is the only one that can be solved in polynomial time; all the others are strongly NP-hard and cannot be solved by exact solution methods in a reasonable computational time [4].

This paper concerns the ST-SR-TA problem, which involves both allocation and scheduling. This scenario is more realistic for mobile robotics, because mobile robots are usually compactly designed to serve a single task at a time, and in most cases, the task only needs one robot to complete it or the task can be decomposed into elements so that a single robot can pick it up. Typical applications of the ST-SR-TA problem include mobile robot swarm surveillance [5,6], factory robot automation [7,8], wireless sensor networks (WSNs) [9], transportation networks [10], etc. Tremendous efforts have been made to solve this problem in the literature, mostly attributing this problem to a vehicle routing problem (VRP) [11,12] or a multiple-traveling-salesman problem (mTSP) [7], then using heuristic or meta-heuristic methods to solve it. However, there exist some challenges that need to be addressed: (1) Heuristics are fast but yield a low solution quality, while meta-heuristics provide a better quality but require significant computational time. (2) Both these methods fail to generate satisfactory solutions in a reasonable computational time for large-scale problems. (3) Most of the works concern only single-depot cases, and few consider another important variant of MRTA—multiple-depot cases—which we think are a prominent configuration, especially when the real-world application is on a larger scale.

It seems hard to balance the solution quality and computation load for conventional methods. Fortunately, the recent success of deep reinforcement learning (DRL) in combination optimization problems casts a new light on MRTA [10]. DRL, as a variant of deep learning, has a powerful high-dimensional representation capability and can learn a strategy from interactions with the environment automatically without any prepared labels; namely, it is unsupervised. Unlike the meta-heuristic counterparts that search the solution space iteratively, the DRL model can output decision-making results in a feed-forward manner, hence taking a short amount of time. Studies show DRL outperforms heuristic- and meta-heuristic-based methods in various combination optimization tasks like traveling salesman problems (TSPs) [13] and job-shop problems (JSPs) [14]. Hu et al. [15] proposed a distributed policy network (DisPN) over a graph and used DRL to solve the famous mTSP. Though quite efficient, the model has to be retrained if the number of agents changes in the application, and the network limits the model to handling single-depot cases only. To overcome this, Cao et al. [16] proposed a decentralized attention neural network (DAN) by modeling the mTSP as a Markov decision process (MDP), resulting in a natural scalability to varying numbers of agents and cities. However, unfortunately, the model is restricted to a single depot too.

Hence, in this paper, we contribute by applying DRL-based methods to MRTA problems from the following three aspects: the network structure is (1) scalable enough to adapt to either variance in the number of robots and tasks or in the paradigm of the depots, i.e., a single depot, multiple depots or even mixed single and multiple depots; (2) light weight, which significantly alleviates the computation load; and (3) robust no matter how the input data are scaled, shifted or rotated. We believe our proposed methods can better solve MRTA in a more general sense.

This paper is organized as illustrated by the research flow chart in Figure 1. Note in Section 2 that market-based and optimization-based approaches are conventional approaches and the DRL-based approach can be seen as the state of the art. Our main contributions lie in Section 4 and are tested through extensive comparative experiments in Section 5. The advantages and disadvantages of the method are addressed in Section 6. Finally, Section 7 concludes the paper and proposes some possible future work.

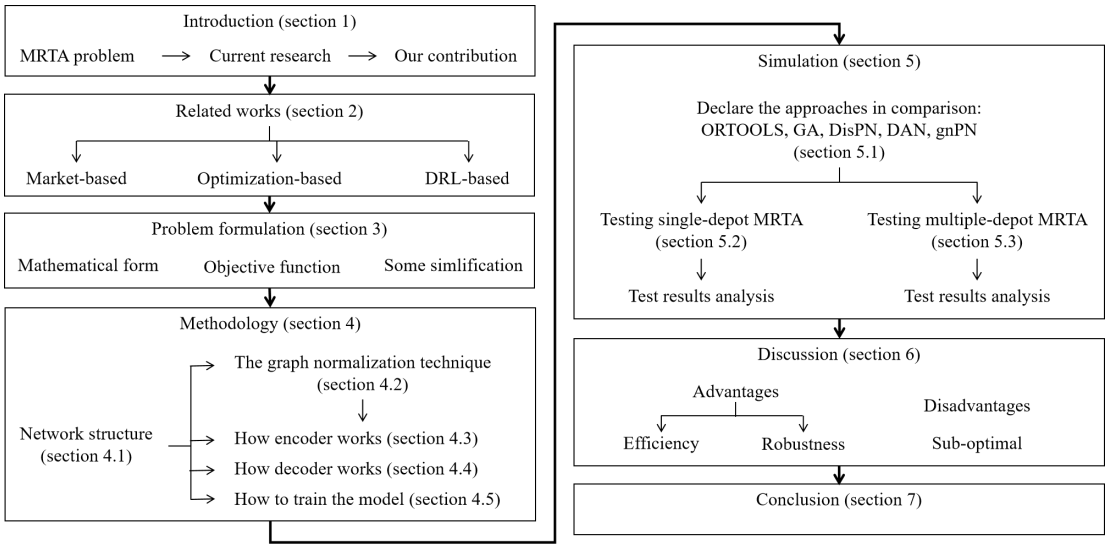

Figure 1. An overview of the paper's structure.

2. Related Works

We first summarize the conventional resolutions to the MRTA problem and then address DRL-based methods in this section.

2.1. Traditional MRTA Solutions

The existing methods of MRTA can be classified into three main categories [17]: (a) market-based, (b) optimization-based, and (c) behavior-based approaches. Market-based approaches are heuristic and inspired by human market trading, where goods are sold to the highest bidder. Similarly, the robots in MRTA bid for tasks according to some deliberately designed rules so that the collective profit can be maximized. A novel market-based approach with look-ahead agents (MALA) is suggested in [18], where each agent first plans a preferred, reward-maximizing tour, and then negotiates and trades with the other agents to get as close as possible to their desired tasks. The authors of [19] presented a similar implementation called Move-and-Improve that comprises four main phases: (1) target allocation, (2) tour construction, (3) negotiation of conflicting targets, and (4) solution improvement, from which the task allocation result is formed incrementally. The auction-bidding framework guarantees a conflict-free solution and the solution quality is shown to be competitive with the genetic algorithm. The consensus-based bundle algorithm (CBBA) [20,21] is deemed one of the most popular auction-based methods; it has shown to be efficient in terms of both computation and communication. It iterates between the bundle construction phase and conflict resolution phase until all the tasks are allocated. However, the authors of [22] address the problems that the CBBA pursues individual profit rather than the collective objective and that bids for a task are highly sensitive to the auction sequence; therefore, they proposed the performance impact (PI) task allocation algorithm, where a significance concept is devised to optimize the global objective directly. Though advantageous regarding a fast computational time and a distributed topology, market-based methods suffer from high failure rates and local minima when compared with optimization-based methods [23].

Optimization-based methods mainly refer to meta-heuristic methods, which make up the majority of the existing works related to MRTA. Task scheduling of multiple unmanned aerial vehicles (UAVs) for urban surveillance is divided into two phases in [5]: a task allocation phase and a single UAV scheduling phase. These phases are iteratively performed until predefined stopping criteria are met. The task allocation stage is realized by the Tabu-list-based simulated annealing (SATL) algorithm, and single UAV scheduling is realized by variable neighborhood descent (VND). Such a divide and conquer diagram can also be found in [7], where a multi-robot car-door spot welding production line is

formulated as a multi-station MRTA problem, and the solution framework is divided into three layers. Based on the genetic algorithm (GA), the three layers of single-robot path planning, multi-robot task assignment, and multi-station task assignment are solved iteratively. The authors of [8] combine the GA and A* algorithms to tackle the problems of task allocation and collision-free path planning of multiple mobile robots carrying out industrial plant inspection tasks, where the GA is used as a task planner to schedule the tasks as well as assign the optimal number of tasks to each robot based on the distance matrix calculated by the A* path planner. Though easy to implement, the GA faces the challenge of balancing between exploration and exploitation. To enhance the diversity of the population and avoid premature convergence of the popular GA, the authors of [24] introduced a beetle antenna search (BAS) mechanism to incorporate with the GA, and the results on the cooperative multi-UAV multiple-stage attack mission scenarios showed a satisfactory performance. Paper [25] formulates the forest firefighting task assignment as an optimization problem and implements a discrete particle swarm optimization (PSO) algorithm to solve it. Ant colony optimization (ACO) is also one of the most popular approaches in MRTA. The authors of [26] combine ACO and the CBBA for the multi-UAV search and rescue problem, where ACO is employed in the inclusion phase to build a bundle of survivors and then possible conflicts in the latter consensus phase are resolved using the CBBA. More meta-heuristic methods can also be found in pure mTSP studies, for example, the artificial beet colony (ABC) and invasive weed optimization (IWO) proposed in [27], the PSO proposed in [28], the shuffled frog-leaping algorithm (SFLA) proposed in [29], the hybrid genetic algorithm (HGA) proposed in [30], and the state transition simulated annealing algorithm (STASA) proposed in [31]. It should be noted that, in some special cases, it is suggested that meta-heuristic methods incorporate some additional operators like 2-opt as a local optimizer to further improve the solution quality [27].

2.2. DRL-Based Methods

In recent years, DRL-based methods have been developed for combinational optimization problems. There are two fundamental motivations: (a) researchers want to replace some heavy calculations in traditional approaches with quick approximation learning and (b) sometimes the algorithm output may be unsatisfactory, so it is desirable to explore the decision space more sufficiently and learn a nonvolatile policy. The most groundbreaking work, to the best of our knowledge, is the pointer network (PN) proposed and studied in [32,33], which consists of an encoder–decoder structure that outputs target classes of the input directly, and, more importantly, the length of the input can be variable. Inspired by the PN, the authors of [13] proposed an attention model to replace the recurrent unit in the PN and trained the model using REINFORCE with a baseline based on a deterministic greedy rollout. Significant improvements were achieved in various traditional combinational problems such as the TSP, the VRP, and the orienteering problem (OP) in their work. The classical TSP using DRL is also studied in [34] by learning an improvement heuristic to guide the selection of the next solution, and in [35] by representing the map as an image and using deep convolutional neural networks (DCNNs) to learn an efficient policy for selecting the next city.

There exist few works related to MRTA or the mTSP. Hu et al. [15] constructed an architecture consisting of a shared graph neural network and distributed policy network (DisPN) to learn a generalized policy capable of outputting near-optimal solutions for the mTSP. The approach is divided into two stages: (a) DRL to learn an allocation of agents to vertices, and (b) OR-TOOLS to resolve the sub-tour planning problem. Though quite efficient, there are two limitations of their model: (a) the architecture fixes the number of agents, which means retraining is needed if the number of agents is to be changed, and (b) it is incapable of assessing multiple-depot cases, despite the fact that a single depot is only a variant of the multiple-depot problem. Another model is the decentralized attention-based neural network (DAN) proposed by Cao et al. [16], where the mTSP is formulated as a sequential decision-making problem: the agent observes the state at each step considering

the potential decisions of other agents, and then casts attention weights to the remaining cities. The advantage of this architecture is that it allows for an arbitrary number of agents compared with [15], whilst the disadvantage is also the single-depot limitation. The authors of [36] also address the single-depot mTSP and propose an attention-based multi-agent reinforcement learning (AMARL) approach that can adapt to varying numbers of agents and cities. It should be noted that a coordinator is mandatory in the architecture to avoid the interaction of agents' simultaneous decision making.

Overall, while the existing DRL-based works made significant contributions to the field of MRTA or the mTSP, there is still a need for further research to address the limitations of existing models, particularly in terms of handling multiple-depot scenarios and accommodating varying numbers of agents.

3. Problem Formulation

The ST-SR-TA MRTA problem will be described by integer linear program (ILP) formulation in this section. Given a set of tasks $T = \{t_1, \ldots, t_n\}$ located in the configuration space (e.g., Cartesian space) and a set of robots $R = \{r_1, \ldots, r_M\}$ in their depots $D = \{d_1, \ldots, d_M\}$, we aim to allocate suitable tasks to the robots, i.e., $X : T \rightarrow R$, in such a way that optimizes the overall system performance subject to a set of constraints. Since the robot is a single task in this paper, each robot has to schedule the tour of the sub-tasks that are allocated to it. The allocation for robot r_k can be represented as $X_k = T|\{x_{kij} = 1\}, \forall k \in \{i, \ldots, M\}, \forall (i,j) \in \{1, \ldots, N\}$, where x_{kij} is the binary decision variable and $x_{kij} = 1$ if robot r_k transits from i to j or $i \neq j$ otherwise $x_{kij} = 0$. N is the total number of tasks and robots, $N = n + M$. For convenience, we view the depots as virtual tasks and combine them as extended tasks $T^e = T \cup D = \{t_1, \ldots, t_n, d_1, \ldots, d_M\}$ so $|T^e| = N$. Suppose the scheduled visiting sequence of robot r_k is Z_k, and since each robot has to depart from its depot and return to the same depot, we have $Z_{k,1} = Z_{k,-1}$. From the point of view of graph theory, the problem can be formally defined on a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. There are costs on E and the costs can be defined using a cost matrix $C = [c_{ij}]^{N \times N}$, whose element c_{ij} represents the cost transiting from node i to j, usually including the travel cost from i to j and the stay time on node j. As we are concerned about an undirected graph herein, C is symmetrical, i.e., $c_{ij} = c_{ji}, \forall i \neq j$. The objective is to minimize the maximum cost (denoted as minmax in most of the literature):

$$X^* = \underset{\{x_{kij}\}}{\operatorname{argmin}} \max(\{\sum_{i=1}^{N}\sum_{j=1}^{N} x_{kij} c_{ij}, \forall k \in \{1, \ldots, M\}\}) \tag{1}$$

or to minimize the average cost of all the robots (denoted as minavg):

$$X^* = \underset{\{x_{kij}\}}{\operatorname{argmin}} \sum_{k=1}^{M}\sum_{i=1}^{N}\sum_{j=1}^{N} x_{kij} c_{ij} / M \tag{2}$$

subject to:

$$\sum_{k=1}^{M}\sum_{j=1}^{n} x_{kij} = 1, \forall i \in \{1, \ldots, n\} \tag{3}$$

$$\sum_{k=1}^{M}\sum_{i=1}^{n} x_{kij} = 1, \forall j \in \{j, \ldots, n\} \tag{4}$$

$$\sum_{j=1}^{n} x_{k(n+k)j} \leq 1, \forall k \in \{1, \ldots, M\} \tag{5}$$

$$\sum_{i=1}^{n} x_{ki(n+k))} \leq 1, \forall k \in \{1, \ldots, M\} \tag{6}$$

$$\sum_{k=1}^{M}\sum_{i=n+1}^{N}\sum_{j=n+1}^{N} x_{kij} = 0 \tag{7}$$

$$\sum_{i\in S}\sum_{j\in S} x_{kij} \le |S| - 1, \forall S \in \{Z_k\}\backslash\{Z_{k,1}\}, S \neq \Phi, \forall k \in \{1,\ldots,M\} \tag{8}$$

where Equations (3) and (4) require that a task (depots not included) must be visited exactly once; constraints (5) and (6) indicate a robot can leave and return to its depot one time or just stay without leaving and returning at all; constraint (7) prohibits unexpected visits of one robot to any other robots' depot; and constraint (8) is known as the sub-tour elimination constraint (SEC) [37] to prevent any unexpected sub-loop inside a robot's tour. Some additional constraints may be added according to the specific application, like the endurance mileage limitation of the robot:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} x_{kij} \le L_k, \forall k \in \{1,\ldots,M\} \tag{9}$$

Practically, the optimization of objective (1) will lead to the shortest possible completion time of the overall tasks, i.e., the make-span, assuming the same moving speed of all the robots, while objective (2) will optimize the mean energy consumption. In practice, objective (1) is more frequently used than objective (2), but we think the standalone objective (1) concerns the max cost only while neglecting the fact that the remaining robots' tours may not be optimized. Hence, we suggest a combination of the two objectives:

$$X^* = \underset{\{x_{kij}\}}{\operatorname{argmin}} \left[\omega_1 \max\left(\{\sum_{i=1}^{N}\sum_{j=1}^{N} x_{kij}c_{ij}, \forall k \in \{1,\ldots,M\}\}\right) + \omega_2 \sum_{k=1}^{M}\sum_{i=1}^{M}\sum_{j=1}^{N} x_{kij}c_{ij}/M\right] \tag{10}$$

$$\omega_1 + \omega_2 = 1 \tag{11}$$

where $\omega_1 > 0$ and $\omega_2 > 0$ are decision preference factors. By setting $\omega_1 \gg \omega_2$, we can obtain a solution that optimizes the make-span time while the energy consumption is still slightly considered.

It should be noted that some simplifications are applied to reduce the input feature complexity of the DRL-based methods in this paper: (a) the cost is defined as the travel cost only, without stay time, and (b) no additional constraints other than (3)–(8) are applied. However, we believe these simplifications can be compensated for by some post-processing technology.

4. Methodology

We resort to DRL-based methods to overcome the drawbacks of conventional heuristic or meta-heuristic approaches. Inspired by [15], our diagram is composed of (a) a policy network that observes the state of all the tasks and robots in the graph and outputs the action of each task to be assigned to each robot, and (b) a route planner to generate the optimal route for each robot. The policy network is also encoder–decoder-structured, but with graph normalization first, which can better extract the node features of the graph and better adapt to configuration variance. The model is trained by leveraging an MRTA simulator without the need to provide predefined labels, i.e., unsupervised, and we use a bootstrapping strategy to promote the policy towards a higher, stable score.

4.1. Policy Network Architecture

Usually, the policy network architecture of DRL comprises an encoder to extract the deep-level features of the input and a decoder to output the action probabilities. With regard to MRTA in this paper, the intuitive input vector would be the raw coordinates of the nodes, like in most related works [15,16,36], but we think this type of input vector contains too much redundant data. In Figure 2, for example, the two graphs are indeed equivalent from

the point of view of the graph configuration, because the right map is the shifted, scaled, and rotated version of the left map, but this will not influence the task allocation result, as MRTA concerns the relative location of each robot. On the one hand, redundant data will make the training process take a long time; on the other hand, the same results cannot be guaranteed under the equivalent configuration.

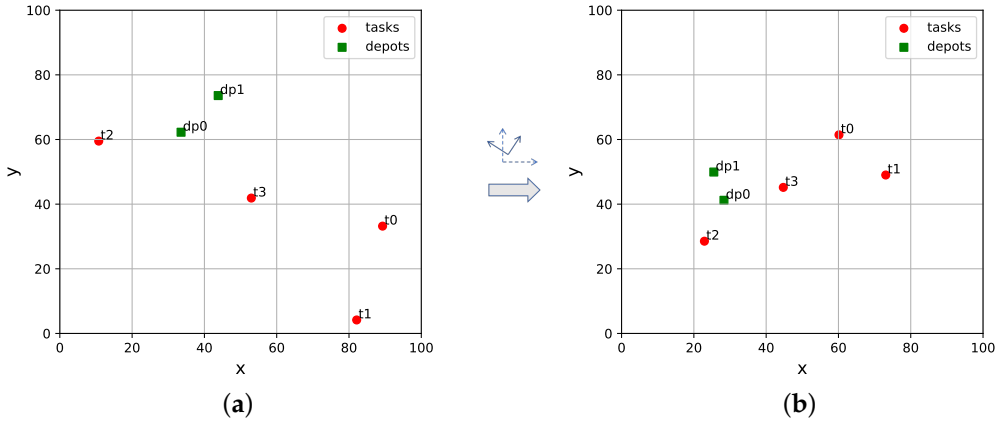

Figure 2. An example of the equivalent configuration of MRTA: (**a**) the original configuration; (**b**) the shifted, scaled, and rotated configuration of (**a**).

Hence, in this paper, we proposed a graph normalization (GNorm) technique to process the input features prior to the encoder, resulting in an architecture shown in Figure 3. Firstly, the input feature is normalized using GNorm, eliminating the effects of shift, scale, and rotation of the configuration. Then, the normalized features are fed into the encoder to capture the node features in the context of the graph. Thirdly, the decoder calculates the action probability that each task is likely to be assigned to each robot, from which the robot can decide which tasks to take. At last, the sub-tour planner schedules the visiting sequence of the sub-tasks for each robot and hence the final output solution. In the following, we will describe the individual units in detail.

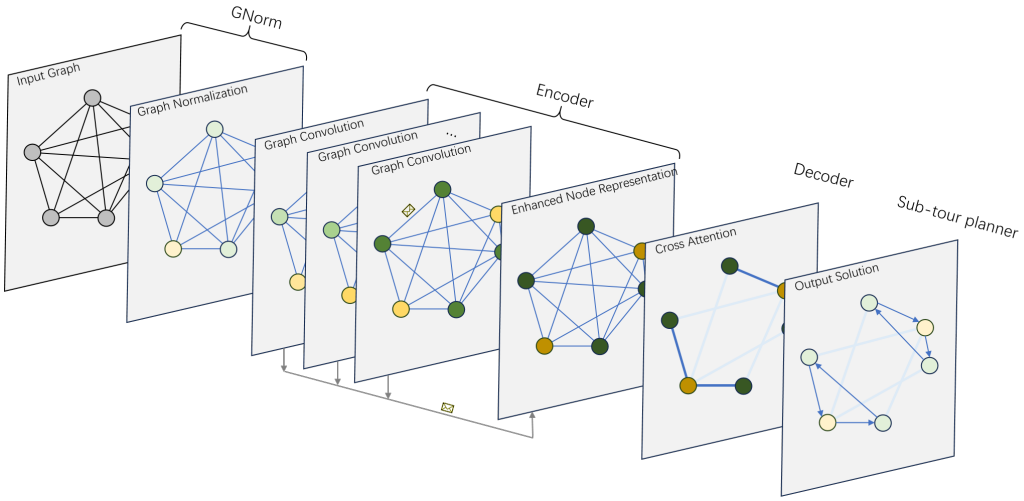

Figure 3. The proposed architecture of MRTA.

4.2. Graph Normalization

The functionality of graph normalization is to transform the raw input vector into a unified input feature to grasp the essence of an input graph.

Let $P = \{p_1, \ldots, p_N\}$ be the extended tasks' (depots included) coordinates. The elements of the cost matrix C can be formulated as:

$$c_{ij} = \|p_j - p_i\|, \forall i, j \in \{1, \ldots, N\} \tag{12}$$

Then, we can find the edge with the maximum cost as the reference for normalization. The two end nodes of the reference edge are:

$$(i^*, j^*) = \underset{i,j}{\operatorname{argmax}} C \tag{13}$$

Since the cost matrix is symmetric, i^*, j^* are impartial indexes. To identify them, we use the aggregation density of the node to decide the exact reference node, i.e.:

$$(i^*, j^*) = \begin{cases} (i^*, j^*), \, if \ \underset{i}{\text{mean}}\, c_{ij} > \underset{j}{\text{mean}}\, c_{ij}, \\ (j^*, i^*), \text{ else.} \end{cases} \tag{14}$$

see Figure 4 for an illustration, where the dashed line is the found reference edge, and the reference nodes are decided as $i^* = 8$, $j^* = 10$ according to (14).

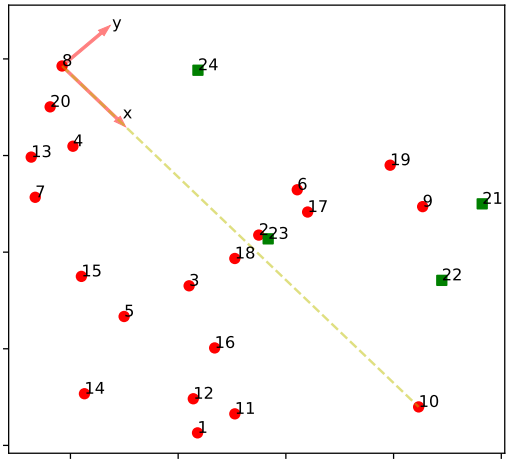

Figure 4. An illustration of deciding the reference edge (dashed line) and the two reference nodes. The red dots and the small green squares are the task nodes and robot nodes, respectively, with their node numbers shown herein.

Based on the reference edge and nodes, all the nodes' coordinates can be normalized:

$$vs. = p_{j^*} - p_{i^*} \tag{15}$$

$$T = \begin{bmatrix} v_x & -v_y \\ v_y & v_x \end{bmatrix} \tag{16}$$

$$\widetilde{p}_i = (p_i - p_{i^*})T / \|p_{j^*} - p_{i^*}\|, \forall i \in \{1, \ldots, N\} \tag{17}$$

where v is the vector of the reference edge and T is the transformation matrix. $\widetilde{p}_i$ is the normalized coordinate of each node; it remains the same whenever a graph is scaled, shifted, or rotated; hence, the essence of the graph is maintained.

However, using $\widetilde{p}_i$ as the input feature is not enough, because when it comes to single-depot MRTA cases, all the depots' features would be the same due to the same raw coordinate, hindering the encoder from discriminating the individual robots. Keep in mind that we want the policy network to work well both for single- and multiple-depot cases.

A remedy is to add a token to the input feature in order to discriminate different robots. We use the normalized ID of the robots as the depots' token, while the tasks' tokens are 0, i.e.:

$$token_i = \begin{cases} 0, 1 \le i \le n, \\ (i-n)/M, n < i \le N. \end{cases} \tag{18}$$

Hence, the normalized input feature proposed in this paper is:

$$u_i = [token_i; \widetilde{p}_i], \forall i \in \{1, \ldots, N\} \tag{19}$$

where $[\cdot;\cdot]$ is the concatenation operation.

4.3. Encoder

The functionality of the encoder is to embed the nodes' features into a unique vector in the high-dimensional configuration space, taking the graph context information into account. We leverage the Graph SAmple and aggreGatE (GraphSAGE) [38] concept to extract the nodes' features in the graph; the detailed process is shown in Figure 5 and is describes as follows.

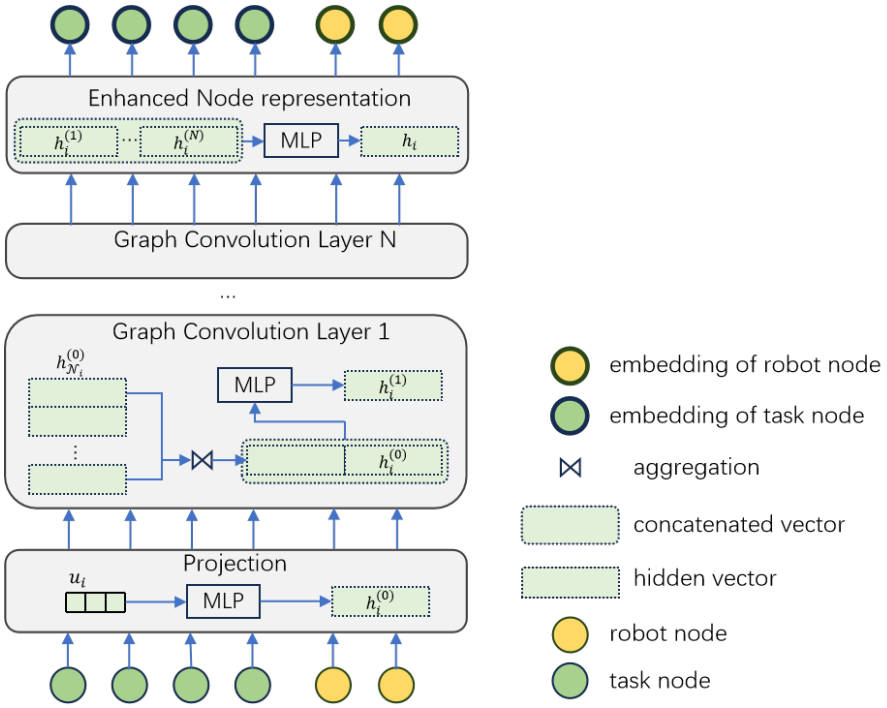

Figure 5. The encoder's working process.

Firstly, the input feature of the node is projected to a higher-dimensional feature in the configuration space:

$$h_i^{(0)} = f(u_i) \tag{20}$$

where $f(\cdot)$ is a multiple-layer perception (MLP) layer with shared and trainable parameters.

Then, the feature vector is fed into graph convolution of N layers to extract the feature of the node in the context of the entire graph. The most important function of the graph convolution layer is to aggregate the neighbors' information that can better represent the node in the graph. For each node i, the message aggregated from its neighbors is:

$$\overline{h}_i^{(t)} = \Phi_{j \in \mathcal{N}_i}(h_j^{(t-1)}) \tag{21}$$

where $\mathcal{N}_i$ denotes the neighbors of node i and $h_j^{(t)}$ is the node representation of node j at the tth graph convolution layer. $\Phi(\cdot)$ is an aggregation function, which must be permutation-independent; we use the mean function herein. The node representation must merge its own message at this step:

$$h_i^{(t)} = \Psi([\bar{h}_i^{(t)}; h_i^{(t-1)}]) \tag{22}$$

where $\Psi(\cdot)$ is an MLP used to map the message to the predefined configuration space. It is preferable that the aggregation process (20)–(22) is repeated three to five times, i.e., $N = 3 \sim 5$, because this would encourage the message to pass farther so that all the nodes can have a better overview of the graph context.

At last, we enhance the node representation by concatenating all the N layers of the graph convolution output and passing over an MLP layer:

$$h_i = \Xi([h_i^{(1)}; \ldots; h_i^{(N)}]) \tag{23}$$

4.4. Decoder

We designed the decoder to output the action probability that each task will be assigned to each robot; this can be realized by a cross-attention mechanism, as illustrated in Figure 6 and the following:

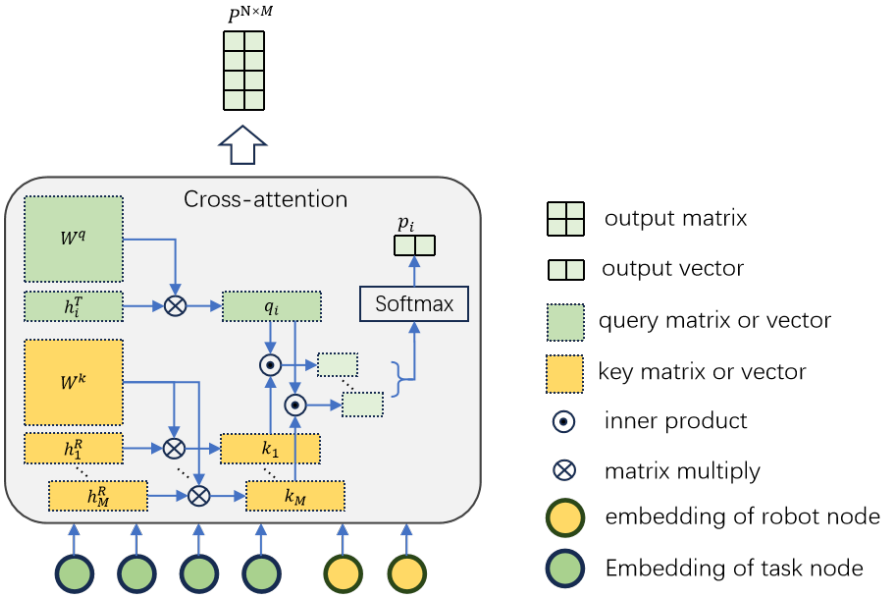

Figure 6. The decoder's working process.

Firstly, the nodes embedding from (24) must be discriminated as tasks and depots. Let us denote the task embedding as h_i^T and the depot embedding as h_j^R. The task node will carry the query vector and the robot node will carry the key vector, computed as follows:

$$q_i = W^q h_i^T, k_j = W^r h_j^R \tag{24}$$

where W^q and W^r are the query and key matrix, respectively, with trainable parameters shared by all the nodes. Then, the task node will cast attention to the robot node, and the attention factor can be calculated by the $softmax$ function:

$$p(i,j) = \frac{e^{q_i \cdot k_j / \sqrt{d_k}}}{\sum_{j=1}^{M} e^{q_i \cdot k_j / \sqrt{d_k}}} \tag{25}$$

where d_k is the dimension of the key vector. The output $p(i, j)$ is actually the probability that task i is likely to be allocated to robot j; it can be sampled by the task node to select a robot, from which the robot will infer the task nodes that are assigned to it.

The last part of the architecture is the visiting order for each robot; this is indeed a TSP problem and many efficient solvers exist to resolve it. Different from the OR-TOOLS employed in [15], we consult Lin–Kernighan–Helsgaun (LKH) solver [39] because it is proven efficient in generating satisfactory solutions while keeping the computational effort at a low level. According to our experience, the LKH solver generates better solutions than OR-TOOLS within 1 s on a TSP scale of less than 100 cities and 5 s on a TSP scale of less than 400 cities.

4.5. Training

The policy network composed of an encoder and a decoder has to be trained before deployment. As reinforcement learning needs no predefined labels, the training process proceeds through an agent interacting with the environment within a simulator. In our simulator, the dataset is generated randomly in a squared map of 1×1; this makes sense because any practical input map can be normalized using the graph normalization process proposed in Section 4.2, resulting in an equivalent configuration in a 1×1 map. The agent observes the status of the environment and outputs the action (the allocation probability in our problem) through the policy network, and a reward will be returned by the environment, from which the agent can update the network parameters in such a way that the reward is maximized. The loss function of the policy is defined as the expected reward of the actions:

$$\mathscr{L}(s;\theta) = \mathbb{E}_{a\sim\pi_\theta(s)} R(a|s) \tag{26}$$

where s is the status, θ are the parameters of the policy network to be learned, a is the action obeying policy π_θ, and $R(a|s)$ is the reward of the action that can be evaluated by the negative of the objective function of (10). The loss function can then be optimized by a policy gradient using the REINFORCE algorithm with the baseline [13]:

$$\nabla_\theta \mathscr{L}(s;\theta) \approx \frac{1}{B}\sum_{i=1}^{B}\sum_{t=1}^{n_i} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})(R(s^{(i)}) - b(s^{(i)})) \tag{27}$$

Note that the loss function is approximated by Monte Carlo sampling in (27), B is the sampling batch size and n_i is the number of actions at the ith sample. Here, baseline $b(s)$ is critical because the baseline can reduce gradient variance and discriminate good and bad actions more clearly, therefore increasing the speed of learning as well as stabilizing the training process. The baseline must improve itself as the training proceeds, so as to bootstrap the reward towards a higher score. In this paper, we simply set the baseline as the reward of the best-so-far model and update it periodically.

Another problem we experienced is that the reward cannot be improved for small-scale instances at the end of the training. This is mainly because the reward is not at a similar amplitude compared with the large-scale instances. We overcome this by replacing the advantage function $R(s^{(i)}) - b(s^{(i)})$ of (27) by a relative advantage:

$$\nabla_\theta \mathscr{L}(s;\theta) \approx \frac{1}{B}\sum_{i=1}^{B}\sum_{t=1}^{n_i} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) \frac{R(s^{(i)}) - b(s^{(i)})}{b(s^{(i)})} \tag{28}$$

The detailed training process is summarized in Algorithm 1. Note that in line 1, the parameter of the baseline model is initialized as the same as the current model and is updated periodically if the performance is improved, as shown in line 14. Line 2 generates a set of evaluation instances used as an auxiliary to judge if the performance is improved or not; line 4 is crucial, where both single- and multiple-depot MRTA instances, with a variable number of robots and tasks, are generated for training, aiming at the generalization capability of the policy network; line 8–10 calculate the rewards of the baseline; line 11–12

update the parameters of policy network by gradient back forward algorithm; and bootstrapping is realized in lines 13–15 by updating the baseline model parameters under certain criteria.

Algorithm 1: Training a policy network with the bootstrapping REINFORCE algorithm for MRTA

Input: min and max number of tasks n_{min}, n_{max}, min and max number of robots m_{min}, m_{max}, sampling batch size B, training epoch E, update interval T, learning rate α.
Output: policy π_θ.

1 Randomly initialize θ; $\theta^b \leftarrow \theta$;
2 $s_{eval} \leftarrow$ Generate a set of evaluation instances;
3 **for** *epoch=0:E* **do**
4 $\quad s \leftarrow$ Generate single- and multiple-depot random instances of size B, each with randomized n and m, $n \in [n_{min}, n_{max}]$, $m \in [m_{min}, m_{max}]$;
5 $\quad p_\theta(\pi|s) \leftarrow$ Policy network feeds forward;
6 $\quad \{a^{(i)}, \log p_\theta(a^{(i)}|s^{(i)})\} \leftarrow$ Greedily sample actions, $\forall i \in \{1, \ldots, B\}$;
7 $\quad \{R^{(i)}\} \leftarrow$ Evaluate episode actions $\{a^{(i)}\}$, $\forall i \in \{1, \ldots, B\}$;
8 $\quad p_{\theta^b}(\pi^b|s) \leftarrow$ Baseline model feeds forward;
9 $\quad \{a^{b(i)}, \log p_{\theta^b}(a^{b(i)}|s^{b(i)})\} \leftarrow$ Greedily sample actions, $\forall i \in \{1, \ldots, B\}$;
10 $\quad \{b^{(i)}\} \leftarrow$ Evaluate actions $\{a^{b(i)}\}$, $\forall i \in \{\forall 1, \ldots, B\}$;
11 $\quad \nabla_\theta \mathscr{L}(s;\theta) \leftarrow$ Policy gradient as per (28);
12 $\quad \theta \leftarrow \theta + \alpha \nabla_\theta \mathscr{L}(s;\theta)$;
13 $\quad$ **if** at every T iterations, mean$[R_{\theta^b}(s_{eval})] >$ mean$[R_\theta(s_{eval})]$ **then**
14 $\quad\quad \theta \leftarrow \theta^b$;
15 $\quad$ **end**
16 **end**

5. Simulation

The advantage of our model is that it works well both for single- and multiple-depot MRTA under variable numbers of robots and tasks, without the need to change the network architecture or additional retraining work. In this section, experiments will be arranged to evaluate the performance of our proposed policy network with graph normalization (denoted as gnPN), compared with heuristic, meta-heuristic, and other DRL-based approaches, from the aspect of both the solution quality and computational time.

5.1. Experimental Set-Up

The single-depot and multiple-depot cases will be studied separately in these experiments. For the single-depot cases, both randomly generated maps and publicly available mTSPLib [40] maps will be studied. Regarding the multiple-depot cases, only randomly generated maps will be present due to the lack of available public datasets.

The approaches use in comparison to ours are:

- A heuristic algorithm, which is realized by OR-TOOLS [41].
- A meta-heuristic algorithm, the GA in this paper.
- Deep-reinforcement-learning-based methods, including the DisPN proposed by Hu. et al. [15] and the DAN proposed by Cao. et al. [16].

The challenge of using the GA to solve MRTA is the chromosome design; here, we used two chromosomes, one to represent the visiting orders and the other to represent breaks, to encode a solution [42]. Since heuristic and meta-heuristic methods run in iteration, we set the stopping criterion of the two as no further improvement in 1000 iterations, and to prevent searching for a long time, we limited the maximum searching time to 180 s.

To ensure a fair comparison, all the approaches were run on the same hardware platform: Intel Core i5 3.0GHz CPU and 16 GB RAM; no GPU is used.

5.2. Testing on Single-Depot MRTA

In single-depot MRTA (denoted as SD-MRTA), all the robots share a common depot. In this part, we generate a certain number of tasks, one common depot and a certain number of robots in a squared map of 1×1. To obtain a stochastic evaluation, we generate 500 instances, then use the five algorithms to solve them independently and calculate an averaged value as the metric. Table 1 shows the comparison results on small- and medium-scale instances, where the solution quality is evaluated by the minmax value and the computational time is indicated by CPU time.

Table 1. Test results of different approaches on small- and medium-scale single-depot MRTA instances; data are averaged over 500 random instances; the best performance for each instance is bold for convenience.

Approaches	$n = 50, m = 5$		$n = 50, m = 10$		$n = 100, m = 5$		$n = 100, m = 10$	
	Minmax	**CPU Time (s)**	**Minmax**	**CPU Time (s)**	**Minmax**	**CPU Time (s)**	**Minmax**	**CPU Time (s)**
OR-TOOLS	**2.121**	0.905	2.026	0.966	**2.414**	9.908	2.195	10.061
GA	2.590	3.624	2.482	3.991	3.246	8.803	2.921	12.257
DisPN [1]	2.143	0.024	1.995	0.027	2.493	0.048	2.135	0.044
DAN [1]	2.314	0.192	2.037	0.240	2.729	0.449	2.181	0.486
gnPN	2.174	**0.001**	**1.955**	**0.001**	2.484	**0.003**	**2.068**	**0.004**

[1] Data slightly different from [15,16], since we re-ran the paper code on our hardware platform rather than the authors' platform.

It can be observed that DRL-based methods produce satisfactory solutions in less computational time than the heuristic and meta-heuristic methods in most cases, and among the DRL-based methods, our proposed gnPN approach performs better than the others. Although the improvement on the minmax value is slight, 1.5%∼3.5% typically, the most exciting enhancement is the inference time; our approach can solve small- and medium-scale problems within 0.01 s, and this can be attributed to the light-weighted architecture of our model.

Scalability or generalization on large-scale problems is an important capability that researchers are interested in for MRTA approaches. Table 2 shows the comparison results on large-scale instances of the five approaches. Again, a significant improvement is observed for our gnPN model, and this improvement is outstanding as the scale of the problem increases. Basically, the inference time is less than 1s for a problem scale of $n \leq 1000$, $m \leq 10$. Note that the DisPN model for different numbers of agents, i.e., variant m, has to be re-trained prior to usage, and the model structure has to be altered if m changes. Both OR-TOOLS and the GA show limited convergence in the predefined time limitation of 180 s. Although the DAN is comparative in terms of solution quality, it suffers from a drastic increase in the computational time with regard to the instance scale.

Furthermore, we can take a close look at specific instances. Fortunately, Necula et al. [40] released public single-depot mTSP datasets marked with (sub)optimal solutions that are, so far, the best. The four datasets we tested in this paper are eil51, berlin52, eil76, and rat99, with the first node assigned as the common depot and the number of agents fixed as $m = 5$. The test results of different approaches on the dataset can be seen in Figure 7. The task allocation of each agent is highlighted by different colors and the individual visiting sequence of an agent is marked with arrows in the figure. OR-TOOLS is the most comparative solution to the optimal so far; even better solutions are found for eil76. The GA performs well on eil51 and berlin52, but deteriorates on eil76 and rat99. As the DisPN model for this experiment is trained on $n = 50, m = 5$, it performs quite well on the eil51; however, the generalization capability is not good enough when it comes to larger-scale

instances. Both the DAN and our gnPN model perform well on larger-scale instances, but the performance of the latter seems more steady. The inference time improvement is indicated in Tables 1 and 2. Another advantage of the gnPN model that can be observed in Figure 7 is that the entanglement across the sub-routs is scarce compared with the other four approaches; this can mainly be attributed to the particularly designed objective function (10) and (11), which mainly optimizes the max tour length while keeping an eye on the tour of the other agents.

Table 2. Test results of different approaches on large-scale single-depot MRTA instances; data are averaged over 500 random instances; the best performance for each instance is bold for convenience.

Approach	$n = 200, m = 5$		$n = 200, m = 8$		$n = 200, m = 10$		$n = 500, m = 5$	
	Minmax	**CPU Time (s)**	**Minmax**	**CPU Time (s)**	**Minmax**	**CPU Time (s)**	**Minmax**	**CPU Time (s)**
OR-TOOLS	**2.919**	139.011	2.647	132.914	2.772	129.144	9.850	180
GA	4.444	22.431	3.855	29.459	3.742	32.542	7.344	83.183
DisPN [1]	3.210	0.201	-	-	2.436	0.161	4.891	0.933
DAN	3.414	1.819	2.628	1.772	2.420	1.786	5.342	15.502
gnPN	3.083	**0.045**	**2.387**	**0.028**	**2.261**	**0.026**	**4.244**	**0.265**
	$n = 500, m = 8$		$n = 500, m = 10$		$n = 800, m = 5$		$n = 800, m = 8$	
	minmax	**CPU time (s)**	**minmax**	**CPU time (s)**	**minmax**	**CPU time (s)**	**minmax**	**CPU time (s)**
	9.918	180	9.923	180	14.787	180	14.869	180
	6.296	99.761	5.879	103.249	10.010	165.935	8.617	174.628
	-	-	3.340	0.537	6.377	2.575	-	-
	3.806	15.576	3.309	15.757	7.327	61.359	5.034	61.096
	3.251	**0.179**	**2.941**	**0.153**	**5.195**	**0.697**	**3.895**	**0.469**
	$n = 800, m = 10$		$n = 1000, m = 5$		$n = 1000, m = 8$		$n = 1000, m = 10$	
	minmax	**CPU time (s)**	**minmax**	**CPU time (s)**	**minmax**	**CPU time (s)**	**minmax**	**CPU time (s)**
	14.793	180	18.364	180	18.368	180	14.853	180
	8.181	180	12.524	180	10.847	180	10.397	180
	4.304	1.406	7.432	4.428	-	-	4.986	2.366
	4.196	61.641	8.751	119.120	5.841	119.117	4.868	119.209
	3.474	**0.390**	**5.810**	**1.128**	**4.282**	**0.729**	**3.831**	**0.639**

[1] For DisPN, the results of $n = *, m = 8$ instances are not available because the model was not trained on this number of agents.

5.3. Testing on Multiple-Depot MRTA

The character of multiple-depot MRTA (denoted as MD-MRTA) is that each robot occupies an independent depot, rather than sharing a common depot like in SD-MRTA. Unfortunately, there is little research on MD-MRTA, but we believe it is worth studying. Furthermore, the multiple-depot scenario is a generalized form of its single-depot counterpart, and, in real applications, the multiple-depot scenario is inevitable if the robot and task scales are large enough.

Because OR-TOOLS and all the existing DRL-based methods cannot be adapted to MD-MRTA cases, we compare our approach with the GA only, and due to the lack of public datasets, we generate instances randomly in this part of the experiment. The test results can be seen in Figure 8. Generally speaking, the GA performs well on instances with $n < 50$, while on larger-scale instances, our model generates better solutions than the GA. Also, refer to Tables 1 and 2 for the computational time enhancement.

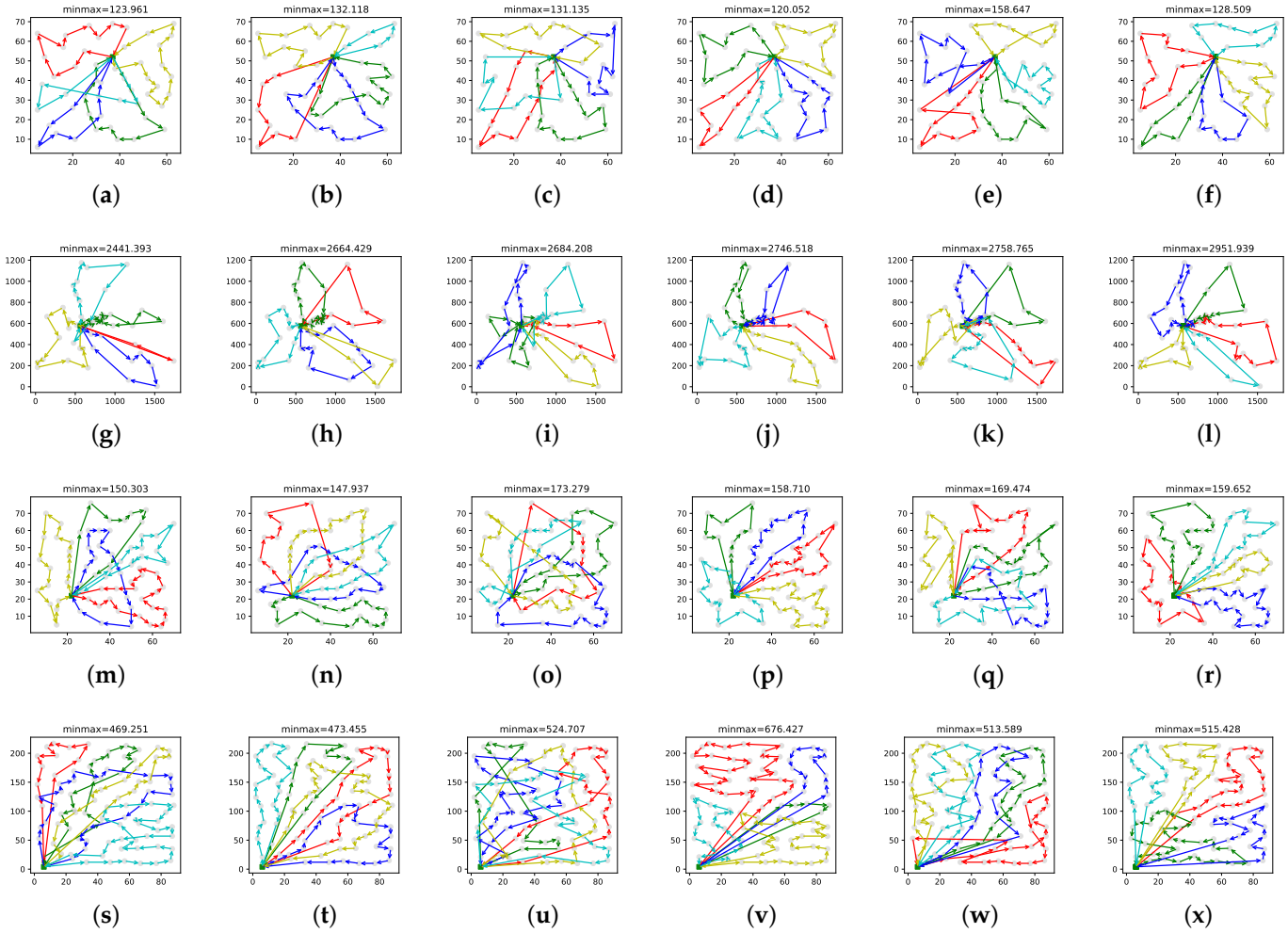

Figure 7. Test results of five different approaches on four public SD-MRTA (the single-depot mTSP) instances with the number of agents fixed as $m = 5$. (**a**–**f**) eil51 instance with the optimal solution so far and solutions found by OR-TOOLS, GA, DisPN, DAN, and gnPN, respectively; (**g**–**l**) berlin52 instance with the optimal solution so far and solutions found by OR-TOOLS, GA, DisPN, DAN, and gnPN, respectively; (**m**–**r**) eil76 instance with the optimal solution so far and solutions found by OR-TOOLS, GA, DisPN, DAN, and gnPN, respectively; (**s**–**x**) rat99 instance with the optimal solution so far and solutions found by OR-TOOLS, GA, DisPN, DAN, and gnPN, respectively.

An interesting yet challenging special form of MRTA is mixed single- and multiple-depot MRTA (MSMD-MRTA); namely, some robots share a common depot like in SD-MRTA while the others have their own independent depots like the scenario of MD-MRTA. This is rarely studied in the literature to the best of our knowledge. But this can be solved easily by our model without any modifications of the network structure or additional re-training work. The comparison result is shown in Figure 9, where more than half of the robots share a common depot, while the others depart from their independent depots. The solutions are also satisfactory for these complex scenarios.

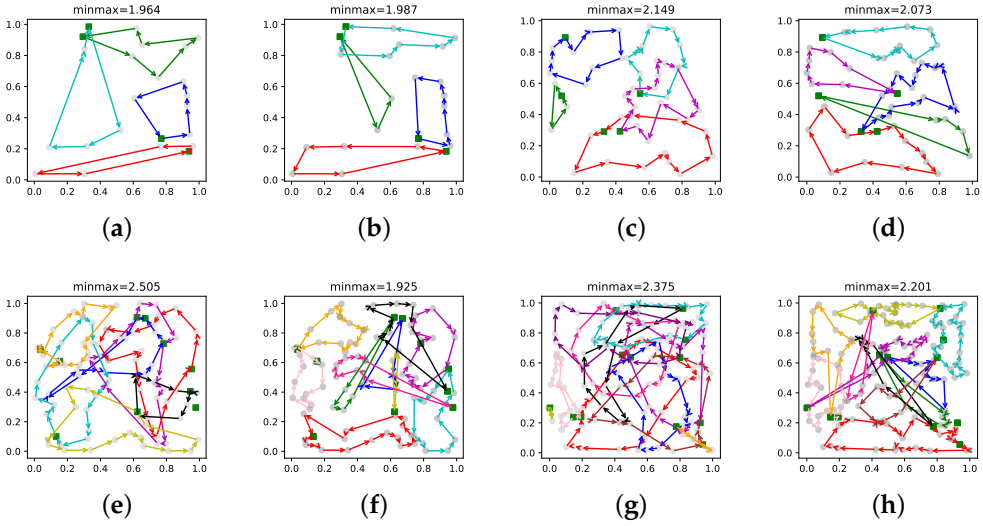

Figure 8. Test result of two different approaches on four MD-MRTA instances. (**a**,**b**) $n = 20$, $m = 4$ instance, solutions found by GA and gnPN, respectively; (**c**,**d**) $n = 50$, $m = 5$ instance, solutions found by GA and gnPN, respectively; (**e**,**f**) $n = 100$, $m = 10$ instance, solutions found by GA and gnPN, respectively; (**g**,**h**) $n = 150$, $m = 12$ instance, solutions found by GA and gnPN, respectively.

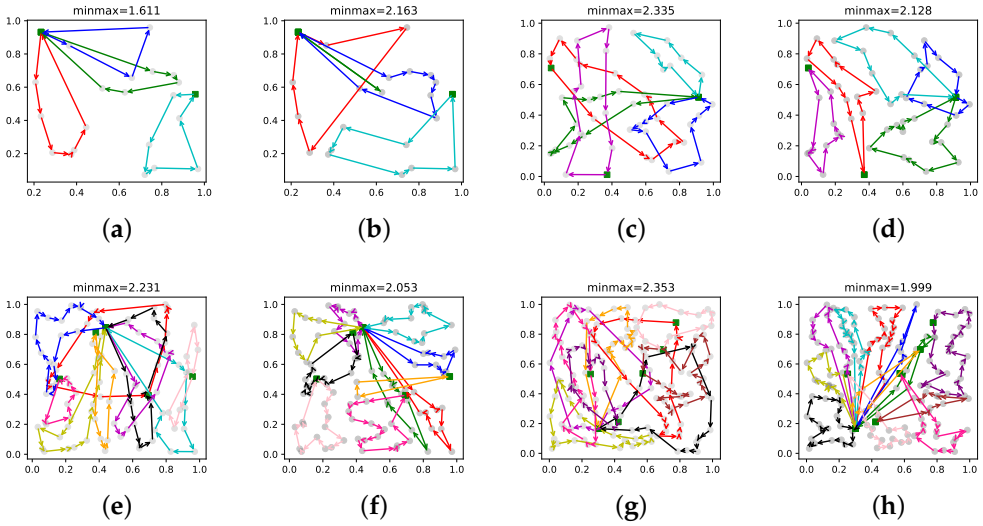

Figure 9. Test result of two different approaches on four MSMD-MRTA instances. (**a**,**b**) $n = 20$, $m = 4$ instance, solutions found by GA and gnPN, respectively; (**c**,**d**) $n = 50$, $m = 5$ instance, solutions found by GA and gnPN, respectively; (**e**,**f**) $n = 100$, $m = 10$ instance, solutions found by GA and gnPN, respectively; (**g**,**h**) $n = 150$, $m = 12$ instance, solutions found by GA and gnPN, respectively.

6. Discussion

The quick feed-forward and learnable characteristics make the proposed policy network architecture an efficient approach to resolving either single-depot, multiple-depot or even mixed single- and multiple-depot MRTA cases, as demonstrated in Section 5. However, the solution quality highly depends on the training data; for example, in Figure 7l, the model performs poorly on the instance of berlin52. The main reason for this is that the training data are generated in a uniform manner during the training process, whilst the tasks in the berlin52 case are not uniformly distributed. To overcome this, a post-processing technique can be applied. For example, Figure 10 shows the test result of the gnPN model post-processed using large neighborhood search (LNS) [43]. There is approx. a 10.9%

improvement in the solution quality; the computational time, however, increased by 2.9 s due to the additional post-processing effort.

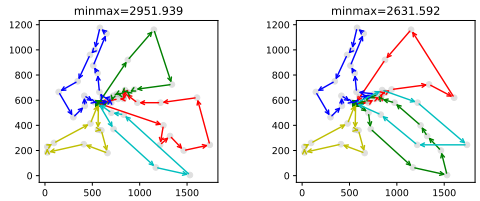

Figure 10. An example of the gnPN model post-processed using large neighborhood search, taking the berlin52 instance as an example. (**a**) Before post-processing, (**b**) after post-processing.

Robustness is inherently guaranteed by our approach thanks to the proposed graph normalization technique. This technique facilitates seamless adaptation to real-world applications, regardless of the map scale or orientation. In contrast, other DRL-based methods struggle with this, as it is hard to decide a coordinate transformation. To illustrate this point, consider Figure 11a,b for the DisPN, where a slight scaling and rotation of the configuration leads to significantly different solutions. In contrast, our approach captures the essential characteristics of the configuration, and the deterministic solution can be guaranteed, see Figure 11c,d.

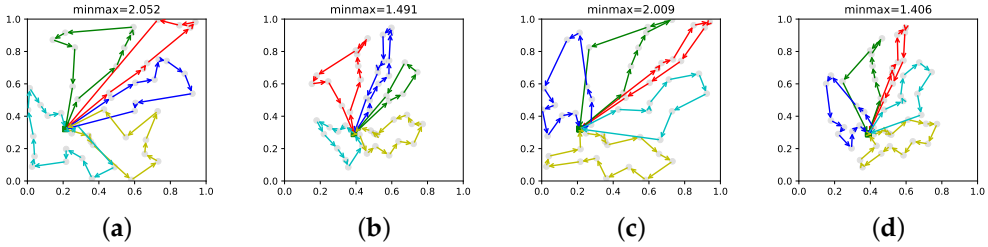

Figure 11. An example to illustrate the functionality of graph normalization. (**a**) DisPN's solution for the original map; (**b**) DisPN's solution for the scaled and rotated map of (**a**); (**c**) gnPN's solution for the same map in (**a**); (**d**) gnPN's solution for the same map in (**b**).

7. Conclusions

Graph deep-reinforcement-learning-based methods are promising to tackle MRTA problems compared with conventional iteration-based methods; satisfactory solutions can be generated within a very short computational time thanks to the feed-forward and trainable policy network. We propose a robust architecture that can handle cases with variable numbers of agents and tasks in single-depot, multiple-depot, or even mixed single- and multiple-depot cases, without modification of the policy network structure and additional re-training efforts. The network is encoder–decoder-structured, using the GraphSAGE concept to extract the nodes' features and the cross-attention mechanism to decode the action. We propose using graph normalization prior to GraphSAGE, which enables an easy adaption to real applications and guarantees a deterministic solution. Extensive experiments demonstrate the high efficiency and robustness of the proposed approach to handle various MRTA scenarios. In addition, the solution quality can be further improved by some post-processing techniques like large neighborhood search.

Future work will pay attention to enhancing the model's capability of handling constraints like time-window and mileage limitations. Attention will also be given to heterogeneous multi-robot task allocation problems, where DRL-based methods can be used to solve the inner task allocation of a group of robots that are still homogeneous.

Author Contributions: Conceptualization, Z.Z., X.J. and Z.Y.; methodology, Z.Z. and X.J.; software, Z.Z.; validation, Z.Z.; formal analysis, X.J. and Z.Y.; investigation, Z.Z.; resources, S.M. and W.S.; data curation, Z.Z.; writing—original draft preparation, Z.Z.; writing—review and editing, X.J.; visualization, Z.Z.; supervision, S.M.; project administration, Z.Y. and J.C.; funding acquisition, S.M. and W.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research is funded by National Natural Science Foundation of China (62303272); Key Research and Development Program of Shandong Province (2021CXGC011304); Project of Natural Science Foundation of Shandong Province (ZR2023QF067, ZR2022QF038, ZR2020MF066); China University Innovation Fund (2021ZYA12004); Qingdao Natural Science Foundation (23-2-1-118-zyyd-jch); Postdoctoral Innovation Project of Shandong Province (SDCX-ZG-202203036); Qingdao Postdoctoral Funding Project (QDBSH20220201013).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: Author Jiyang Chen was employed by the company Shandong Zhengzhong Information Technology Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DAN	Decentralized Attention Neural Network
DisPN	Distributed Policy Network
DRL	Deep Reinforcement Learning
GA	Genetic Algorithm
GraphSAGE	Graph SAmple and aggreGatE
LKH	Lin–Kernighan–Helsgaun
MD-MRTA	Multiple-Depot Multi-Robot Task Allocation
MLP	Multiple Layer Perception
MRS	Multi-Robot System
MSMD-MRTA	Mixed Single- and Multiple-Depot Multi-Robot Task Allocation
MRTA	Multi-Robot Task Allocation
mTSP	Multiple Traveling Salesman Problem
SD-MRTA	Single-Depot Multi-Robot Task Allocation

References

1. Verma, J.K.; Ranga, V. Multi-robot coordination analysis, taxonomy, challenges and future scope. *J. Intell. Robot. Syst.* **2021**, *102*, 10. [CrossRef] [PubMed]
2. Khamis, A.; Hussein, A.; Elmogy, A. Multi-robot task allocation: A review of the state-of-the-art. In *Cooperative Robots and Sensor Networks 2015*; Springer: Cham, Switzerland, 2015; pp. 31–51.
3. Gerkey, B.P.; Matarić, M.J. A formal analysis and taxonomy of task allocation in multi-robot systems. *Int. J. Robot. Res.* **2004**, *23*, 939–954. [CrossRef]
4. Korsah, G.A.; Stentz, A.; Dias, M.B. A comprehensive taxonomy for multi-robot task allocation. *Int. J. Robot. Res.* **2013**, *32*, 1495–1512. [CrossRef]
5. Liu, H.; Li, X.; Wu, G.; Fan, M.; Wang, R.; Gao, L.; Pedrycz, W. An iterative two-phase optimization method based on divide and conquer framework for integrated scheduling of multiple UAVs. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 5926–5938. [CrossRef]
6. Liu, Y.; Song, R.; Bucknall, R.; Zhang, X. Intelligent multi-task allocation and planning for multiple unmanned surface vehicles (USVs) using self-organising maps and fast marching method. *Inf. Sci.* **2019**, *496*, 180–197. [CrossRef]
7. Zhou, B.; Zhou, R.; Gan, Y.; Fang, F.; Mao, Y. Multi-robot multi-station cooperative spot welding task allocation based on stepwise optimization: An industrial case study. *Robot. Comput.-Integr. Manuf.* **2022**, *73*, 102197. [CrossRef]
8. Jose, K.; Pratihar, D.K. Task allocation and collision-free path planning of centralized multi-robots system for industrial plant inspection using heuristic methods. *Robot. Auton. Syst.* **2016**, *80*, 34–42. [CrossRef]
9. Zheng, H.; Yuan, J. An Integrated Mission Planning Framework for Sensor Allocation and Path Planning of Heterogeneous Multi-UAV Systems. *Sensors* **2021**, *21*, 3557. [CrossRef]
10. Wang, Q.; Tang, C. Deep reinforcement learning for transportation network combinatorial optimization: A survey. *Knowl.-Based Syst.* **2021**, *233*, 107526. [CrossRef]

11. Mahmud, M.S.A.; Abidin, M.S.Z.; Buyamin, S.; Emmanuel, A.A.; Hasan, H.S. Multi-objective route planning for underwater cleaning robot in water reservoir tank. *J. Intell. Robot. Syst.* **2021**, *101*, 9. [CrossRef]
12. Yan, M.; Yuan, H.; Xu, J.; Yu, Y.; Jin, L. Task allocation and route planning of multiple UAVs in a marine environment based on an improved particle swarm optimization algorithm. *EURASIP J. Adv. Signal Process.* **2021**, *2021*, 94. [CrossRef]
13. Kool, W.; Van Hoof, H.; Welling, M. Attention, learn to solve routing problems! In Proceedings of the 7th International Conference on Learning Representations. ICLR, New Orleans, LA, USA, 6–9 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–25.
14. Wang, L.; Hu, X.; Wang, Y.; Xu, S.; Ma, S.; Yang, K.; Liu, Z.; Wang, W. Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Comput. Netw.* **2021**, *190*, 107969. [CrossRef]
15. Hu, Y.; Yao, Y.; Lee, W.S. A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs. *Knowl.-Based Syst.* **2020**, *204*, 106244. [CrossRef]
16. Cao, Y.; Sun, Z.; Sartoretti, G. Dan: Decentralized attention-based neural network to solve the minmax multiple traveling salesman problem. *arXiv* **2021**, arXiv:2109.04205.
17. Chakraa, H.; Guérin, F.; Leclercq, E.; Lefebvre, D. Optimization techniques for Multi-Robot Task Allocation problems: Review on the state-of-the-art. *Robot. Auton. Syst.* **2023**, *168*, 104492. [CrossRef]
18. Karmani, R.K.; Latvala, T.; Agha, G. On scaling multi-agent task reallocation using market-based approach. In Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007), Cambridge, MA, USA, 9–11 July 2007; IEEE: Piscataway, NJ, USA, 2007; pp. 173–182.
19. Koubâa, A.; Cheikhrouhou, O.; Bennaceur, H.; Sriti, M.F.; Javed, Y.; Ammar, A. Move and improve: A market-based mechanism for the multiple depot multiple travelling salesmen problem. *J. Intell. Robot. Syst.* **2017**, *85*, 307–330. [CrossRef]
20. Choi, H.L.; Brunet, L.; How, J.P. Consensus-based decentralized auctions for robust task allocation. *IEEE Trans. Robot.* **2009**, *25*, 912–926. [CrossRef]
21. Brunet, L.; Choi, H.L.; How, J. Consensus-based auction approaches for decentralized task assignment. In Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit, Honolulu, HI, USA, 18–21 August 2008; p. 6839.
22. Zhao, W.; Meng, Q.; Chung, P.W. A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario. *IEEE Trans. Cybern.* **2015**, *46*, 902–915. [CrossRef] [PubMed]
23. Geng, N.; Meng, Q.; Gong, D.; Chung, P.W. How good are distributed allocation algorithms for solving urban search and rescue problems? A comparative study with centralized algorithms. *IEEE Trans. Autom. Sci. Eng.* **2018**, *16*, 478–485. [CrossRef]
24. Wang, Z.; Wang, B.; Wei, Y.; Liu, P.; Zhang, L. Cooperative multi-task assignment of multiple UAVs with improved genetic algorithm based on beetle antennae search. In Proceedings of the 2020 39th Chinese Control Conference (CCC), Shenyang, China, 27–29 July 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1605–1610.
25. Chen, K.; Sun, Q.; Zhou, A.; Wang, S. Adaptive multiple task assignments for uavs using discrete particle swarm optimization. In Proceedings of the Internet of Vehicles. Technologies and Services Towards Smart City: 5th International Conference, IOV 2018, Paris, France, 20–22 November 2018; Proceedings 5; Springer: Berlin/Heidelberg, Germany, 2018; pp. 220–229.
26. Zitouni, F.; Harous, S.; Maamri, R. A distributed approach to the multi-robot task allocation problem using the consensus-based bundle algorithm and ant colony system. *IEEE Access* **2020**, *8*, 27479–27494. [CrossRef]
27. Venkatesh, P.; Singh, A. Two metaheuristic approaches for the multiple traveling salesperson problem. *Appl. Soft Comput.* **2015**, *26*, 74–89. [CrossRef]
28. Zhou, H.; Song, M.; Pedrycz, W. A comparative study of improved GA and PSO in solving multiple traveling salesmen problem. *Appl. Soft Comput.* **2018**, *64*, 564–580. [CrossRef]
29. Dong, Y.; Wu, Q.; Wen, J. An improved shuffled frog-leaping algorithm for the minmax multiple traveling salesman problem. *Neural Comput. Appl.* **2021**, *33*, 17057–17069. [CrossRef]
30. Mahmoudinazlou, S.; Kwon, C. A hybrid genetic algorithm for the min–max Multiple Traveling Salesman Problem. *Comput. Oper. Res.* **2024**, *162*, 106455. [CrossRef]
31. Zhang, Y.; Han, X.; Dong, Y.; Xie, J.; Xie, G.; Xu, X. A novel state transition simulated annealing algorithm for the multiple traveling salesmen problem. *J. Supercomput.* **2021**, *77*, 11827–11852. [CrossRef]
32. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 2692–2700.
33. Bello, I.; Pham, H.; Le, Q.V.; Norouzi, M.; Bengio, S. Neural combinatorial optimization with reinforcement learning. In Proceedings of the International Conference on Machine Learning (Workshop), Sydney, Australia, 6–11 August 2017.
34. Wu, Y.; Song, W.; Cao, Z.; Zhang, J.; Lim, A. Learning improvement heuristics for solving routing problems. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *33*, 5057–5069. [CrossRef] [PubMed]
35. Ling, Z.; Zhang, Y.; Chen, X. A Deep Reinforcement Learning Based Real-Time Solution Policy for the Traveling Salesman Problem. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 5871–5882. [CrossRef]
36. Gao, H.; Zhou, X.; Xu, X.; Lan, Y.; Xiao, Y. AMARL: An Attention-Based Multiagent Reinforcement Learning Approach to the Min-Max Multiple Traveling Salesmen Problem. *IEEE Trans. Neural Netw. Learn. Syst.* 2023, *early access*. [CrossRef]
37. Bektas, T. The multiple traveling salesman problem: An overview of formulations and solution procedures. *Omega* **2006**, *34*, 209–219. [CrossRef]
38. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1024–1034.

39. Helsgaun, K. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *Eur. J. Oper. Res.* **2000**, *126*, 106–130. [CrossRef]
40. Necula, R.; Breaban, M.; Raschip, M. Tackling the bi-criteria facet of multiple traveling salesman problem with ant colony systems. In Proceedings of the 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), Vietri sul Mare, Italy, 9–11 November 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 873–880.
41. Perron, L.; Furnon, V. ORTOOLS. 2020. Available online: https://developers.google.com/optimization/ (accessed on 16 April 2024).
42. Shuai, Y.; Yunfeng, S.; Kai, Z. An effective method for solving multiple travelling salesman problem based on NSGA-II. *Syst. Sci. Control Eng.* **2019**, *7*, 108–116. [CrossRef]
43. Pisinger, D.; Ropke, S. *Handbook of Metaheuristics*; Springer: Berlin/Heidelberg, Germany, 2019.