SplitNet: A Reinforcement Learning Based Sequence Splitting Method for the MinMax Multiple Travelling Salesman Problem

Hebin Liang[1*], Yi Ma[1*], Zilin Cao[1], Tianyang Liu[1], Fei Ni[1], Zhigang Li[1], Jianye Hao[1,2†]

[1] College of Intelligence and Computing, Tianjin University
[2] Noah's Ark Lab, Huawei
{lianghebin, mayi, zilin24, liutianyang, fei_ni, scs_lzg, jianye.hao}@tju.edu.cn

Abstract

MinMax Multiple Travelling Salesman Problem (mTSP) is an important class of combinatorial optimization problems with many practical applications, of which the goal is to minimize the longest tour of all vehicles. Due to its high computational complexity, existing methods for solving this problem cannot obtain a solution of satisfactory quality with fast speed, especially when the scale of the problem is large. In this paper, we propose a learning-based method named SplitNet to transform the single TSP solutions into the MinMax mTSP solutions of the same instances. Specifically, we generate single TSP solution sequences and split them into mTSP subsequences using an attention-based model trained by reinforcement learning. We also design a decision region for the splitting policy, which significantly reduces the policy action space on instances of various scales and thus improves the generalization ability of SplitNet. The experimental results show that SplitNet generalizes well and outperforms existing learning-based baselines and Google OR-Tools on widely-used random datasets of different scales and public datasets with fast solving speed.

Introduction

Travelling Salesman Problem (TSP) is a classical NP-Hard problem. Given a list of nodes and the distances between each pair of nodes, a vehicle is required to find the shortest route that starts from a specified origin node, visits each node exactly once, and returns to the origin node (Jünger, Reinelt, and Rinaldi 1995). As an important variant of TSP, the multiple Travelling Salesman Problem (mTSP) requires multiple vehicles to visit every node under the same constraints of TSP, making the problem more complicated. Compared with TSP of only one vehicle (single TSP), mTSP is more widely applicable in practical scenarios. MinMax is a typical optimization objective of mTSP aiming to minimize the longest travelling distance of all vehicles. It is of more practical significance since it is related to balancing the workload, which is important in various scheduling and routing problems. For example, in rescue applications, it's desired to find people in danger as soon as possible by dispatching humans and vehicles efficiently. In the scenario of exploration of natural resources, the equitable distribution of workload for each vehicle is important.

After decades of development, many methods for routing problems have emerged both in OR and ML communities. Conventional methods for solving MinMax mTSP mainly resort to various heuristic algorithms. Popular commercial solvers include Gurobi (Gurobi 2020) and CPLEX (IBM 2018). Evolutionary algorithm, also called genetic algorithm, is also used to solve mTSP (Carter and Ragsdale 2006). Lin-kernighan-helsgaun solver (LKH) (Helsgaun 2000), an advanced heuristic solver, implements an outstanding local search heuristic that benefits from the variable depth search. LKH3 (Helsgaun 2017), the extension of LKH, is proposed to solve TSP with additional constraints such as mTSP and can obtain approximated optimal or even optimal results on many benchmark instances. OR-Tools (Google 2012) tends to seek approximated optimal solutions on large-scale mTSP with relatively fast speed. (Lupoaie et al. 2019) proposed an algorithm SOM based on meta-heuristic algorithms and is combined with self-organizing map. Most of these conventional methods either compute expensively or perform terribly when solving MinMax mTSP, especially when the scale of problems is enormous. In addition, it cannot be ignored that these conventional heuristic algorithms usually rely on domain knowledge and the rules should be manually constructed, which could be of limited generalizability.

With the development of deep Reinforcement Learning (RL), many methods for solving routing problems based on deep learning have been proposed in recent years. (Bello et al. 2016; Kool, Van Hoof, and Welling 2018; Xin et al. 2021a; Gao et al. 2020; Chen and Tian 2019; Li, Yan, and Wu 2021; Xin et al. 2021b; Duan et al. 2020; Fu, Qiu, and Zha 2021; Kwon et al. 2020; Zong et al. 2022; Kool et al. 2022; Vidal 2022) are examples that use learning methods to solve VRP, a typical routing problem. As for the MinMax mTSP, to the best of our knowledge, there are three RL-based algorithms. (Hu, Yao, and Lee 2020) designed a clustering method to solve the MinMax mTSP. It assigns nodes to different clusters with distributed policy networks, and uses meta-heuristic methods to obtain the single TSP solutions of each cluster. (Cao, Sun, and Sartoretti 2021) proposed a constructive policy, i.e., Decentralized Attention-based Neural Network (DAN) based on Trans-

[*]Equal contribution. † Corresponding author.

former, and trained the model using RL. (Park, Bakhtiyar, and Park 2021) introduced type-aware graph attention on the basis of DAN and proposed a new constructive method, which can efficiently extract the features in the graph. However, although these RL-based methods achieve faster speed and stronger generalization ability compared with traditional methods, there is still a certain gap in solution quality.

Considering the similarity in the topology of solutions of a single TSP and mTSP, we investigate the relationship between the optimal solutions of the two problems on the same instances and find that they have $63\% \sim 74\%$ overlap. Therefore, an intuitive idea is to directly split a single TSP sequence and connect the exposed end nodes with the origin node to get the mTSP solution. Motivated by this, we propose a method named SplitNet that transforms a single TSP solution into a MinMax mTSP solution using a splitting operation to obtain a high-quality solution. The workflow of SplitNet includes the following three steps: 1) **Generate**: generate single TSP solution sequences; 2) **Split**: split each single TSP solution sequence and connect the exposed end nodes with the origin node to get several subsequences, the number of which equals to the number of vehicles; 3) **Reform**: reform each subsequence to further get the optimized MinMax mTSP solution. To enhance the generalized performance on large-scale MinMax mTSP, we further design a decision region to reduce the scope of SplitNet's splitting operation. Experiments show that SplitNet can achieve better performance than several baselines on random and public datasets with relatively faster speed.

Our main contributions are as follows:

(1) Based on the high similarity between the optimal solutions of single TSP and mTSP of the same instance, we for the first time propose the framework of transforming single TSP solution sequences into MinMax mTSP solution sequences.

(2) Under the above framework, we design a single TSP solution sequence splitting model and the corresponding RL training scheme. We also design a decision region to reduce the splitting action space to further improve the generalization of the splitting model.

(3) Compared with learning-based methods and Google OR-Tools, our algorithm achieves superior performance on public dataset mTSPLib (Necula, Breaban, and Raschip 2015) and various scales of random datasets with faster speed.

Problem Formulation

In this section, we introduce the formulation and optimization objective of the MinMax mTSP. Figure 1 shows an mTSP instance. On a 2-dimensional Euclidean plane, assume there are N nodes, including one origin node n_0 and $N-1$ other nodes denoted as $n_1, ..., n_{N-1}$. m vehicles start from the same origin node to visit other nodes and return to the origin node. In the following, unless otherwise claimed, we regard N as the number of total nodes (including the origin node) and m as the number of total vehicles. An mTSP solution consisting of m subsequences is denoted as $[..., [n_0, a_{i1}, ..., a_{ij}, ..., a_{iM(i)}, n_0], ...]$, where

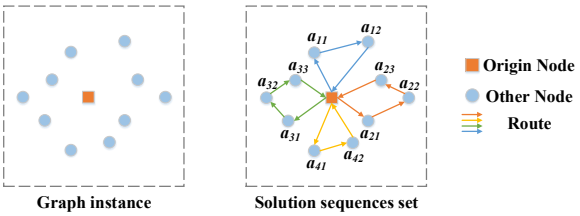

Figure 1: An mTSP instance. Each instance is composed of one origin node and several other nodes $a_{11}, ..., a_{42}$. The arrows of different colors represent the routes in different subsequences.

$a_{ij} \in \{n_1, ..., n_{N-1}\}, 1 \le i \le m, 1 \le j \le M(i)$, $M(i)$ indicates the total number of visited nodes except for n_0 of the i-th subsequence, and $\Sigma_{i=1}^{m} M(i) = N-1$. Define $d(n_i, n_j)$ as the Euclidean distance between two nodes n_i and n_j, then the total length of the i-th $(1 \le i \le m)$ subsequence is:

$$\mathcal{L}_i = d(n_0, a_{i1}) + \sum_{j=1}^{M(i)-1} d(a_{ij}, a_{i(j+1)}) + d(a_{iM(i)}, n_0) \tag{1}$$

The optimization objective is to minimize the maximum travelling length of all the vehicles:

$$\min \max\{\mathcal{L}_i | \forall i, 1 \le i \le m\} \tag{2}$$

subject to

$$a_{ij_1} \neq a_{ij_2}, 1 \le i \le m, 1 \le j_1 < j_2 \le M(i) \tag{3}$$

$$A_i \cap A_j = \emptyset, 1 \le i < j \le m \tag{4}$$

$$\bigcup_{i=1}^{m} A_i = V \tag{5}$$

where $A_i = \{a_{i1}, ..., a_{iM(i)}\}$ represents the set of all nodes in the i-th subsequence except for the origin node, $V = \{n_1, ..., n_{N-1}\}$ represents the set of all nodes in the instance except for the origin node. Constraint 3 ensures that any node in the mTSP solution sequence except for the origin node is only visited once by a certain vehicle. Constraint 4 ensures that any node in the mTSP solution sequence except for the origin node is only visited by one vehicle. Constraint 5 ensures that each node except for the origin node is included in the mTSP solution sequence.

Method

In the initial research phase, we find that the overlap rate of the optimal solution for single TSP and the optimal solution for mTSP of the same instance exceeds 63%. Therefore, an intuitive idea is to directly split an optimal single TSP solution and connect the exposed end nodes with the origin node to get a satisfactory MinMax mTSP solution.

Motivated by this, in this section, we design a method named SplitNet to transform single TSP solutions into MinMax mTSP solutions. The workflow of SplitNet includes the following three steps, as shown in Figure 2: 1) **Generate**: generate several single TSP solution sequences; 2) **Split**: split each single TSP solution sequence into m segments and connect the exposed end nodes with the origin

node to transform each single TSP solution sequence into a MinMax mTSP solution consisting of m subsequences; 3) **Reform**: reform each subsequence to get the optimized MinMax mTSP solution. In the following subsections, we introduce them in detail.

Generate Single TSP Solution Sequences

As it cannot be guaranteed to obtain the optimal solution of the MinMax mTSP starting from an optimal single TSP solution sequence after the above splitting and reforming operations, we sample several different single TSP solution sequences instead of only using an optimal one to obtain a higher quality solution of the MinMax mTSP. Specifically, for a given instance, we first find k nodes closest to the origin node, and then designate these nodes as the first nodes to visit after the vehicle leaves the origin node. On this basis, we use LKH3 to generate k different single TSP solution sequences. Then the splitting and reforming operations are performed on each of the k sequences as described in the following sections.

Split Single TSP Solution Sequences

Given a single TSP solution sequence, with the purpose of minimizing the maximum travelling length of all the m vehicles, we perform $m-1$ splitting operations to transform it into MinMax mTSP solutions with m subsequences. This process can be achieved using exhaustive algorithms, but its time complexity, i.e., $O(N^{m-1})$, makes it impossible to solve the splitting problem in an acceptable time. Therefore, we model the decision-making process that successively selects and splits edges from the entire single TSP solution sequence as a Markov Decision Process (MDP). The edge splitting is performed by a solution sequence splitting policy $\boldsymbol{\pi}$ (called SplitNet policy), which is instantiated as a neural network model designed based on Attention Model (AM) (Kool, Van Hoof, and Welling 2018) and trained using RL. We also keep track of the obtained policies during training and denote $\boldsymbol{\pi}_b$ as the rollout baseline policy that achieves the best results in the validation set among all the obtained policies at different time steps.

Markov Decision Process (MDP) We first introduce the MDP modeling of solution splitting, including the state, action, and reward.

State. The state consists of each edge's static and dynamic features in the single TSP solution sequence. The static features of an edge include the length of the edge and the distances from the two end nodes of the edge to the origin node. The dynamic features include which edges can be split, the number of remaining time steps, and the length of the longest subsequence of all the already obtained subsequences. All the above features are normalized and concatenated together.

Action. We define the action of the RL agent as a splitting operation. A splitting operation is to choose an edge from the solution sequence, split the edge and connect the two end nodes with the origin node, as shown in Figure 2. For a single TSP solution $[a_0, a_1, ..., a_i, a_{i+1}, ..., a_{N-1}, a_N]$ (in particular $a_0 = a_N = n_0$), we define e_i as the edge connecting a_i and a_{i+1}. When splitting e_i, if $a_i \neq n_0$ and $a_{i+1} \neq n_0$, then we get two subsequences $[n_0, a_1, ..., a_i, n_0]$ and $[n_0, a_{i+1}, ..., a_{N-1}, n_0]$; if $a_i = n_0$, we get $[n_0]$ and $[n_0, a_{i+1}, ..., a_{N-1}, n_0]$; if $a_{i+1} = n_0$, we get $[n_0, a_1, ..., a_i, n_0]$ and $[n_0]$.

In practice, we force the agent to split the edges in the order of $e_0 \sim e_{N-1}$ to make the training process faster and more stable. We stipulate that when the agent has split the edge e_i, it cannot split any edge in $\{e_0, ..., e_{i-1}\}$. In other words, these edges are masked in the subsequent time steps.

Reward. Intuitively, we can directly set the negative number of the longest sub-tour length as the reward to match our optimization objective. However, similar to the consideration of ScheduleNet (Park, Bakhtiyar, and Park 2021), the scale of the reward should neither depend on the problem size (N, m), nor on the quality of single TSP solutions, but only on the solution qualities gap between the current RL policy $\boldsymbol{\pi}$ and the rollout baseline policy $\boldsymbol{\pi}_b$. We set the expectation of cost under the current policy, i.e., the tour length of the longest subsequence, to $\mathcal{L}(\boldsymbol{\pi})$. We also set the cost of $\boldsymbol{\pi}_b$ to $\mathcal{L}(\boldsymbol{\pi}_b)$, thus the solution qualities gap is $\mathcal{L}(\boldsymbol{\pi}) - \mathcal{L}(\boldsymbol{\pi}_b)$. Considering that our model needs to be trained on instances of different scales and the scale of the gap of the solution qualities is also different, we further use $\mathcal{L}(\boldsymbol{\pi}_b)$ to normalize $\mathcal{L}(\boldsymbol{\pi}) - \mathcal{L}(\boldsymbol{\pi}_b)$. Therefore, our reward function is as follows:

$$r_t = \begin{cases} 0, t \neq T, \\ -\dfrac{\mathcal{L}(\boldsymbol{\pi}) - \mathcal{L}(\boldsymbol{\pi}_b)}{\mathcal{L}(\boldsymbol{\pi}_b)}, t = T. \end{cases} \tag{6}$$

where t is the index of the current state, and T is the index of the terminal state. This reward formulation is effective in reducing variance and accelerating the coverage speed of RL algorithm (Williams 1992). Note that this reward function is sparse, which means the non-zero reward can only be obtained when the splitting is finished.

Model of SplitNet Policy In this section, we introduce the model structure of our splitting policy as shown in Figure 3.

Encoder. The input of the encoder includes static and dynamic features of each edge, and the output is the embedding of each edge. The static edge features are passed through linear projection (LP) and L attention blocks (AB) to extract the static embeddings of edges based on their different importance in the single TSP solution sequence. The static embeddings are calculated only once in the entire decision-making process. In contrast, the dynamic information such as which edges can be split and how many time steps are left is updated after each splitting. We use an MLP to extract the dynamic embedding of each edge. Then the static information and the dynamic information are merged to get the mixed embedding of each edge. Note that we also introduce the positional encoding used by (Vaswani et al. 2017) into the static edge embeddings so that the model can perceive the serialization features of the ordered input.

Decoder. The decoder takes a context $c = [h_{graph}, h_{last}]$ as input, where h_{graph} is the mean pooling of all edge embeddings and h_{last} is the embedding of the last selected edge. The context is input into a Multi-Head Attention (MHA) layer as a query, while the mixed embedding of all

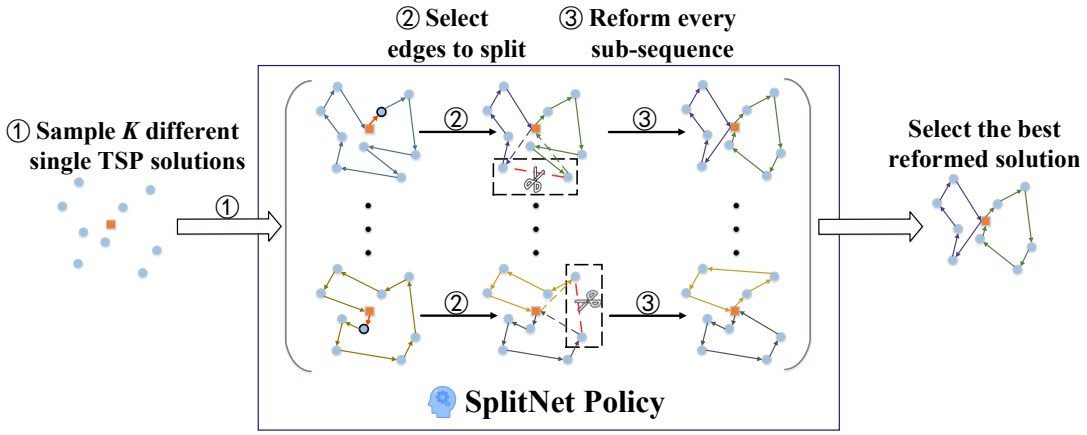

Figure 2: SplitNet Framework. The red dotted line represents the edge to split, and the dotted arrows represent the edges formed by connecting the two end nodes of the split edge to the origin node.

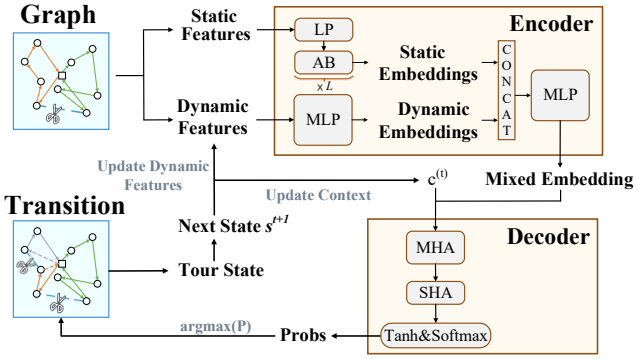

Figure 3: Model of SplitNet Policy

edges is processed by an MLP to obtain the key and value, respectively. The output of the MHA layer is then input into a single head attention (SHA) layer as a query, together with the key calculated based on the mixed embedding of all edges. The SHA layer outputs N single values for each edge. Then these values are limited to [-10, 10] by a tanh activation function, and the values of unavailable edges are masked to negative infinity. Finally, a softmax layer is used to obtain the probability of selecting each edge to split.

Decision Region However, the performance can be catastrophically poor when directly transferring the SplitNet policy introduced above to large-scale mTSP after training on small-scale datasets. This phenomenon is aroused by the dramatic change in the size of the policy action space on large-scale problems compared to small-scale problems. One solution is to restrict the range of edges that the policy can split at each time step so that its action spaces on problems of different scales become similar, thereby improving its generalization.

Since our goal is to minimize the longest subsequence, ideally, the length of each subsequence obtained by dividing a single TSP sequence and connecting with the origin node should be relatively similar, i.e., the ideal ratio of the length of each subsequence to the entire TSP sequence should be the same. To this end, we can approximate the ideal ratio of the length of the new subsequence obtained via splitting to the length of the remaining sequence after the splitting operation at the i-th splitting using:

$$\text{Ratio}_i = \begin{cases} \dfrac{\frac{S_0}{m} + \bar{l}}{(m-1)\frac{S_0}{m} + \bar{l}}, i = 0, \\ \dfrac{\frac{S_0}{m} + 2\bar{l}}{(m-1-i)\frac{S_0}{m} + \bar{l}}, 1 \le i \le m-2. \end{cases} \tag{7}$$

where S_0 is the total length of the single TSP solution sequence, $\bar{l}$ is the average distance from all nodes to the origin node.

At the i-th splitting, for all the remaining $N - i$ edges, we calculate the ratio of the length of the new subsequence obtained via splitting each edge to the length of the remaining sequence after the splitting operation, respectively. Then we can find the edge whose corresponding ratio is closest to the approximated ideal ratio, and we name it as the core edge. Considering the difference between the approximated ideal ratio Ratio_i and the real ideal ratio, directly splitting the core edge may yield sub-optimal solutions. Therefore, to help the policy find the edge that can yield optimal solutions, we delineate the neighbourhood of the core edge and let the policy learn to select and split edges within this neighbourhood. We name the neighbourhood of the core edge as the decision region, i.e., the reduced action space, where the ideal splitting edges locate. An example is given in Figure 4. Through experiments, we also verify that edges that yield high-quality subsequences after splitting are often located at adjacent positions in the sequence at each decision step. It means restricting candidate edges to be split by specifying the decision region may have a very limited negative impact on finding the optimal solutions.

Training of SplitNet Policy We train the SplitNet policy with the REINFORCE (Williams 1992) method with rollout baseline. Given an mTSP instance s, its loss function is as

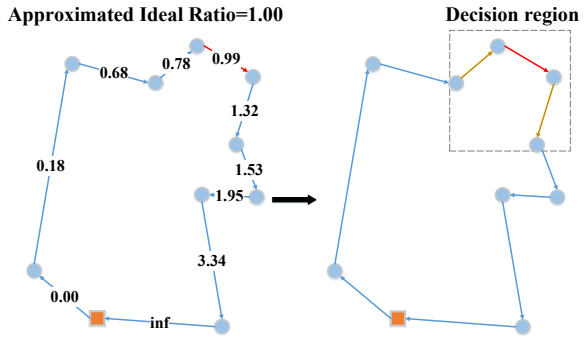

Figure 4: The process of obtaining the decision region. The number marked on each edge represents its corresponding ratio value. As the ratio of the red arrow is closest to the approximated ideal ratio, it is set to the core edge. Centered on it, the edges in the gray dotted square box together constitute the decision region. The policy then chooses to split an edge within the decision region.

follows:

$$\nabla\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\boldsymbol{\pi}|s)}[G\nabla\log p_\theta(\boldsymbol{\pi}|s)] \tag{8}$$

where G is the return obtained after finishing all the splitting, $p_\theta(\pi|s)$ is the probability distribution defined by SplitNet policy for a given instance s, from which we can sample a splitting action. As the reward function described in the MDP subsection is a sparse reward, G is equivalent to r_T. By substituting r_T in Equation 6 to Equation 8, the above loss function can be rewritten:

$$\nabla\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\boldsymbol{\pi}|s)}[\frac{\mathcal{L}(\boldsymbol{\pi}) - \mathcal{L}(\boldsymbol{\pi}_b)}{\mathcal{L}(\boldsymbol{\pi}_b)}\nabla\log p_\theta(\boldsymbol{\pi}|s)] \tag{9}$$

Reform Subsequences

The above splitting method will produce m high-quality subsequences with similar lengths. However, the obtained subsequences may still be different from the optimal ones. After splitting an edge, we obtain two end nodes. Directly connecting one of the end nodes to the origin node is equivalent to forcing the corresponding vehicle to either visit the end node firstly after leaving the origin node, or to visit it lastly before returning to the origin node. It's obvious that the nodes visiting order can still be further optimized.

Therefore, to further improve the solution quality of mTSP, we perform a reforming operation for each subsequence. The specific method is to reform each subsequence by taking the solution obtained by splitting as the initial solution of LKH3 for further optimization. The reforming can eliminate the unreasonable edges generated by splitting and effectively improve the quality of each subsequence. Besides, considering that the nodes of the instances have been relatively evenly divided into m parts, the size of each subproblem is relatively small. Therefore, the reforming will only take a very short time.

Experiments

In order to verify the effectiveness of our framework, we trained SplitNet policy on random datasets generated uniformly in the unit square $[0, 1]^2$ with the number of nodes ranging from 30 to 100, and tested it on random datasets with the number of nodes ranging from 30 to 1000 (we generate 100 instances for each problem scale) and the public benchmark mTSPLib (Necula, Breaban, and Raschip 2015). We generate 30720 random uniform instances as the training set and generate 25 different single TSP solution sequences for each instance using the generating method described in the method section. The number of vehicles in each training instance is randomly generated independently in each epoch, ranging from 3 to 15. We train the model for 80 epochs, of each the gradient step is 1920, and the batch size is 400. During the inference of splitting edges, we sample 64 splitting policies (denoted as s.64) instead of using the greedy method (denoted as g.), and the best sampled policy is adopted as described in (Kool, Van Hoof, and Welling 2018). All the tests are running on a single A100 GPU with Intel(R) Xeon(R) CPU @ 2.20GHz.

In order to obtain the rollout baseline as described before, we generate a validation set containing 320 instances and 8000 solution sequences to validate all policies obtained during training. The training and validation set share the same distribution of the number of nodes and vehicles, but the number of vehicles in each instance is fixed when generating the validation sets for fair comparison.

Note that although our research focuses on the MinMax mTSP, it is proved that our framework also has the potential for solving MinSum mTSP. We run the exhaustive splitting algorithm to solve the small-scale Capacitated Vehicle Routing Problem(CVRP, a typical MinSum mTSP) and achieve better results than OR-Tools and learning-based constructive approaches.

Complete experimental results are provided in appendix.

Baselines

When testing on random uniform datasets, we compare our algorithm with LKH3, OR-Tools, and two problem-specific learning-based methods GNN-DisPN (Hu, Yao, and Lee 2020) and DAN (Cao, Sun, and Sartoretti 2021). We also modify the state-of-the-art learning-based method L2D (Learn to delegate) (Li, Yan, and Wu 2021) for MinSum CVRP (a method iteratively improves the solution by identifying subproblems and delegating their improvement to a subsolver) to apply it to MinMax mTSP. The results of baselines in Table 1 and 2 are reproduced using open-sourced codes from the official implementations. When testing on the mTSPLib datasets, we compare the results of our algorithm with OR-Tools, LKH3, and learning-based algorithms, including (Hu, Yao, and Lee 2020; Cao, Sun, and Sartoretti 2021; Park, Bakhtiyar, and Park 2021; Li, Yan, and Wu 2021). Because of the consistency of the datasets, in Table 4, we directly quote the results of OR-Tools, DAN, and ScheduleNet reported by (Park, Bakhtiyar, and Park 2021), while the results of LKH3, GNN-DisPN, and L2D are reproduced using open-sourced codes.

Results on Random MTSP Datasets

We tested the algorithms on the random datasets of various scales. Table 1 reports the results on small-scale ran-

N	30			50				100				200			
m	2	3	5	2	5	7	10	2	5	10	15	2	10	15	20
LKH3	2.71	2.17	1.91	3.17	2.01	1.93	1.92	4.08	2.22	1.97	1.97	5.50	2.05	2.03	2.03
OR-Tools	2.86	2.26	1.95	3.40	2.07	2.00	2.00	4.52	2.40	2.26	2.29	6.23	2.79	2.92	2.90
GNN-DisPN	3.22	2.63	2.01	3.58	2.14	2.10	1.99	5.11	2.56	2.22	2.04	7.37	2.97	2.30	2.15
DAN(g.)	3.46	2.59	2.10	4.19	2.28	2.10	2.02	5.49	2.70	2.17	2.10	7.39	2.42	2.23	2.18
DAN(s.64)	3.01	2.31	1.95	3.73	2.12	1.98	1.94	5.03	2.52	2.06	2.01	7.17	2.31	2.13	2.09
L2D	3.17	2.47	2.16	3.69	2.28	2.09	1.98	4.66	2.56	2.10	2.02	6.04	2.26	2.12	2.07
SplitNet(s.64)	**2.72**	**2.23**	**1.94**	**3.25**	**2.08**	**1.95**	**1.91**	**4.17**	**2.42**	**2.01**	**1.97**	**5.61**	2.27	**2.12**	**2.07**

Table 1: Random mTSP results.

dom datasets, of which the number of nodes ranges from 30 to 200 and the number of vehicles ranges from 2 to 20. It is noted that SplitNet improves the results by about 10% compared with OR-Tools and reaches performance closer to LKH3. Table 2 reports the results on large-scale mTSP datasets. The number of nodes ranges from 400 to 1000, and the number of vehicles is set to 3 and 10 for comprehensive comparison. It should be emphasized that although SplitNet is trained on random datasets with the number of nodes ranging from 30 to 100, it obtains much better results than OR-Tools and learning-based methods DAN and GNN-DisPN on the large-scale datasets. It also reaches slightly better results than L2D with faster speed. The results demonstrate the superior generalization ability of SplitNet on large-scale MinMax mTSP.

Table 3 reports the average computation time of each algorithm on instances of different scales without considering the I/O cost. The computation time of SplitNet is the sum of the computation time of all steps described in the method section. As we can see, the solving speed of SplitNet is faster than that of L2D, DAN, OR-Tools, and LKH3, especially when encountering large-scale mTSP. Note that techniques such as multi-threading sampling can be used to further improve the speed of SplitNet.

Results on Public Benchmarks

We also test the performance of SplitNet on real-world mTSPLib datasets (Necula, Breaban, and Raschip 2015) after training on the random datasets. MTSPLib contains four datasets namely Eil51, Berlin52, Eil76, and Rat99, of which the corresponding number of nodes are 51, 52, 76, and 99. The number of vehicles in each group is set to 2, 3, 5, and 7. The results of different datasets in Table 4 indicate that SplitNet has the least gap with CPLEX/LKH3 and outperforms OR-Tools, GNN-DisPN, DAN, ScheduleNet and L2D in most instances. By examining the performance of SplitNet on real-world datasets, we can further confirm its outstanding generalization ability. Note that the performance of CPLEX in Table 4 is inferior to that of LKH3 as the solution of CPLEX is the upper bound of the optimal solution while LKH3 can often find the optimal solution.

Ablation Studies

Effectiveness of generating multiple single TSP solution sequences As mentioned before, it cannot be guaranteed to obtain the optimal solution of MinMax mTSP from the optimal single TSP solution by splitting and reforming operations. Therefore, we sample multiple single TSP solution sequences before feeding them into the SplitNet policy to improve the quality of the obtained mTSP solution.

We sample 1, 10, 20, 30, and 40 single TSP solution sequences using LKH3 and compare the results of using different numbers of sampled sequences. As shown in Table 5, the more sequences we sample, the better the results of mTSP solution will become. When the sampling number k is set to 30, compared with the results of k setting to 1, the length can be reduced by around $0.6\% \sim 5.8\%$. This indicates that the sampling techniques can be beneficial to improving the policy performance. When the sampling number $k = 40$, the policy performance improvement is limited while the computation time could be increased compared with that when $k = 30$. Therefore, to balance the performance and the computation time of our algorithm, we set $k = 30$ in our sampling process.

Improvements brought by the decision region In order to prove that the decision region can improve the generalization ability of SplitNet policy, we train the model with the length of decision region (denoted as len) of $\{10, 20, 40, 60, 80\}$ and compare them with two special situations: (1) the model without using decision region, recorded as $len = inf$; (2) the model directly taking the centers of decision regions as actions, recorded as $len = 1$. To compare the generalization of different models, we train them on the datasets with N ranging from 30 to 100, and test them on random datasets with $N = 200, 400, 600$.

As we can see from Table 6, the models using decision region can achieve better results on large-scale problems than the model without decision region and the model taking the centers of decision regions as actions. This indicates that introducing decision region can significantly improve the generalization of the SplitNet policy. We can also conclude that SplitNet can achieve better performance on datasets of different numbers of nodes when the length of decision region is set to 10 or 20.

Benefits of Reforming subsequences To verify the effectiveness of the reforming operation, we record the maximum lengths of the subsequences from the initial output of the SplitNet policy, and compare them with that of the reformed subsequences using LKH3 (i.e. the final output of our algorithm). The results in Table 7 indicate that the maximum lengths of the subsequences are reduced averagely by around $0.8\% \sim 3.1\%$ after reforming.

N	400		600		800		1000	
m	3	10	3	10	3	10	3	10
LKH3	5.22	2.20	6.27	2.39	7.18	2.60	7.99	2.82
OR-Tools(1800s)	6.15	5.24	9.81	9.79	11.74	11.76	14.92	15.22
GNN-DisPN	11.10	7.75	13.41	10.84	16.63	13.09	18.84	14.63
DAN(g.)	7.08	2.96	9.43	3.62	11.05	4.20	13.25	4.80
DAN(s.64)	6.78	2.81	9.12	3.46	10.73	4.07	12.99	4.69
L2D	5.57	2.52	6.55	2.75	7.41	2.93	8.20	3.18
SplitNet(s.64)	**5.40**	2.59	**6.38**	2.82	**7.27**	3.07	**8.06**	3.32

Table 2: Results on the large-scale random mTSP datasets. '1800s' means the running time of OR-Tools is upper limited by 1800 seconds.

N	30		50			100			200			400	1000
m	3	5	5	7	10	5	10	15	10	15	20	10	10
LKH3	4.8	2.4	11.0	11.1	9.5	30.2	32.4	25.6	133.8	112.7	109.5	484.3	920.0
OR-Tools	0.3	0.4	2.0	2.0	2.0	26.2	26.9	24.8	209.7	214.4	207.6	2522.0	>30000
DAN(s.64)	17.5	17.6	27.4	31.7	36.6	55.7	62.2	75.1	98.4	106.5	148.4	222.6	422.2
L2D	4.8	2.5	6.0	4.4	2.2	15.1	7.7	4.4	15.7	11.5	6.7	42.2	131.5
SplitNet(s.64)	0.4	0.6	**1.1**	**1.3**	**1.6**	**2.7**	**3.5**	**4.2**	**6.5**	**7.8**	8.6	**14.3**	**55.1**

Table 3: Comparison of computation times (in seconds).

Instance	Eil51				Berlin52				Rat99			
m	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	728.8	587.2	469.3	443.9
LKH3	222.7	159.6	118.1	112.1	4127.9	3189.5	2440.9	2440.9	666.0	517.7	454.1	438.6
OR-Tools	243.3	170.5	127.5	112.1	4665.5	3311.3	2482.6	2440.9	762.2	552.1	473.7	442.5
GNN-DisPN	260.0	202.5	120.1	120.8	5081.5	4431.8	2855.2	2724.7	1067.8	881.1	854.4	641.4
DAN(g.)	274.2	178.9	158.6	118.1	5226.0	4278.4	2758.8	2696.8	930.8	674.1	504.0	466.4
DAN(s.64)	252.9	178.9	128.2	114.3	5097.7	3455.7	2677.1	2494.5	966.5	697.7	495.6	462.0
ScheduleNet(g)	263.9	200.5	131.7	116.9	4826.1	3644.2	2757.8	2514.6	843.8	671.8	524.3	480.8
ScheduleNet(s.64)	239.3	173.5	125.8	112.2	4591.6	3276.1	2517.3	2441.4	781.2	627.1	502.3	464.4
L2D	257.3	181.4	130.3	120.1	4179.8	4103.6	2865.2	2669.5	786.6	547.9	537.2	452.5
SplitNet(s.64)	**228.3**	**164.7**	127.5	112.6	4244.2	**3191.2**	2662.9	**2440.9**	**683.3**	577.8	**491.3**	458.9

Table 4: Results on mTSPLib. The symbol * refers to the optimal result.

N	50			100		
m	5	7	10	5	10	15
k=1	2.182	1.987	1.918	2.569	2.058	1.984
k=10	2.105	1.952	1.909	2.453	2.020	1.974
k=20	2.090	1.947	1.907	2.434	2.015	1.972
k=30	2.082	1.945	1.907	2.420	2.009	1.972
k=40	2.078	1.942	1.908	2.407	2.010	1.972

Table 5: Results of generating k initial single TSP sequences.

len \ N	200	400	600
1	2.32	2.66	2.89
10	**2.25**	2.61	2.85
20	2.27	**2.59**	**2.82**
40	2.37	2.67	2.86
60	2.46	2.78	2.92
80	2.50	2.87	3.04
inf	2.62	3.62	4.16

Table 6: Results of various decision regions.

N	30		50		100	
m	3	5	5	7	5	10
init	2.299	1.962	2.124	1.960	2.498	2.027
Reform	2.232	1.942	2.082	1.945	2.420	2.009

Table 7: Effectiveness of reforming the subsequences.

Conclusions

In this paper, we propose SplitNet to transform the single TSP solutions into the MinMax mTSP solutions of the same instances. We generate several single TSP solution sequences and split them into mTSP subsequences using an attention-based model trained by reinforcement learning, coupled with a reforming technique. To further improve the generalization ability, we design a decision region to significantly reduce the action space of the splitting policy. The experimental results show that SplitNet can achieve superior performances compared with existing baselines on different scales of random datasets and public datasets with a fast solving speed. We leave the adaptation of SplitNet to more complicated fields as our future work.

Acknowledgements

The work is supported by the National Natural Science Foundation of China (Grant Nos.: U1836214), and the new Generation of Artificial Intelligence Science and Technology Major Project of Tianjin under grant: 19ZXZNGX00010.

References

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Cao, Y.; Sun, Z.; and Sartoretti, G. 2021. DAN: Decentralized Attention-based Neural Network to Solve the MinMax Multiple Traveling Salesman Problem. *arXiv preprint arXiv:2109.04205*.

Carter, A. E.; and Ragsdale, C. T. 2006. A new approach to solving the multiple traveling salesperson problem using genetic algorithms. *European journal of operational research*, 175(1): 246–257.

Chen, X.; and Tian, Y. 2019. Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 32.

Duan, L.; Zhan, Y.; Hu, H.; Gong, Y.; Wei, J.; Zhang, X.; and Xu, Y. 2020. Efficiently solving the practical vehicle routing problem: A novel joint learning approach. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 3054–3063.

Fu, Z.-H.; Qiu, K.-B.; and Zha, H. 2021. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 7474–7482.

Gao, L.; Chen, M.; Chen, Q.; Luo, G.; Zhu, N.; and Liu, Z. 2020. Learn to design the heuristics for vehicle routing problem. *arXiv preprint arXiv:2002.08539*.

Google. 2012. OR Tools. https://developers.google.com/optimization. Accessed: 2022-8-1.

Gurobi. 2020. Gurobi optimizer reference manual. https://www.gurobi.com/. Accessed: 2022-8-1.

Helsgaun, K. 2000. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European journal of operational research*, 126(1): 106–130.

Helsgaun, K. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 24–50.

Hu, Y.; Yao, Y.; and Lee, W. S. 2020. A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs. *Knowledge-Based Systems*, 204: 106244.

IBM. 2018. CPLEX Optimizer. https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer. Accessed: 2022-8-1.

Jünger, M.; Reinelt, G.; and Rinaldi, G. 1995. The traveling salesman problem. *Handbooks in operations research and management science*, 7: 225–330.

Kool, W.; van Hoof, H.; Gromicho, J.; and Welling, M. 2022. Deep policy dynamic programming for vehicle routing problems. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 190–213. Springer.

Kool, W.; Van Hoof, H.; and Welling, M. 2018. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*.

Kwon, Y.-D.; Choo, J.; Kim, B.; Yoon, I.; Gwon, Y.; and Min, S. 2020. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 21188–21198.

Li, S.; Yan, Z.; and Wu, C. 2021. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34: 26198–26211.

Lupoaie, V.-I.; Chili, I.-A.; Breaban, M. E.; and Raschip, M. 2019. SOM-guided evolutionary search for solving minmax multiple-TSP. In *2019 IEEE congress on evolutionary computation (CEC)*, 73–80. IEEE.

Necula, R.; Breaban, M.; and Raschip, M. 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)*, 873–880. IEEE.

Park, J.; Bakhtiyar, S.; and Park, J. 2021. ScheduleNet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv preprint arXiv:2106.03051*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Vidal, T. 2022. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140: 105643.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3): 229–256.

Xin, L.; Song, W.; Cao, Z.; and Zhang, J. 2021a. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *Proceedings of 35th AAAI Conference on Artificial Intelligence*, 12042–12049.

Xin, L.; Song, W.; Cao, Z.; and Zhang, J. 2021b. NeuroLKH: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems*, 34: 7472–7483.

Zong, Z.; Wang, H.; Wang, J.; Zheng, M.; and Li, Y. 2022. RBG: Hierarchically Solving Large-Scale Routing Problems in Logistic Systems via Reinforcement Learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 4648–4658.