THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 23, Number 1/2022, pp. 79–88

A SAMPLING ALGORITHM BASED ON SUPERVISED LEARNING FOR A DYNAMIC TRAVELING SALESMAN PROBLEM

Jiahao JIANG, Jian GAO

Dalian Maritime University, College of Information Science and Technology, Dalian, China
Corresponding author: Jian GAO, E-mail: gaojian@dlmu.edu.cn

Abstract. The Traveling Salesman Problem (TSP) plays an important role in modern logistics, and heuristic algorithms were intensively investigated to solve the problem and its variants. The TSP is always dynamic in real-life applications, and a static TSP cannot be used to model dynamic cases. Therefore, in this paper, we consider a TSP with dynamic customers and propose a sampling algorithm. To solve the dynamic problem, we transform a TSP solution to actions, and propose a novel supervised learning mechanism with a Multilayer Perceptron (MLP) to predict the actions in real time. We also propose to encode travel states and train the model by sampling dynamic cases. Thereafter, a sampling algorithm based on the trained model is proposed to make decisions so as to deal with online dynamic requests. We carried out extensive experiments on TSP instances. From the results, we conclude that the proposed approach outperforms existing TSP heuristic algorithms including 2-OPT, LKH, and Google OR-Tools. We also show the average runtime of our method for producing a path is better than comparative algorithms.

Key words: Traveling salesman problem, supervised learning, combinatorial optimization problem, sampling algorithm.

1. INTRODUCTION

The Traveling Salesman Problem (TSP) is a fundamental combinatorial optimization problem in both artificial intelligence community and operations research community [1]. Given a set of customers (nodes) and costs of all pairs of nodes, the task of solving the TSP is to find a tour with minimum cost ensuring each node is visited exactly once. The TSP is one of the most widely studied problems and is widely used in logistics and manufacturing [2, 3]. Solving a TSP has to find the optimal solution. The TSP is easy to describe, but it is quite difficult to solve. The problem is NP-hard in the general case, because the Hamiltonian cycle problem is proved to be NP-complete [4].

There have been many publications on studying the classical TSP and various algorithms to solve this problem have been proposed. To solve the TSP exactly, we always use integer programming solvers or constraint programming solvers [5]. It is likely that the runtime for those exact algorithms increases exponentially with the number of nodes, and they cannot be solved in a reasonable computation time for large-scale problems. Since exact approaches usually fail to deal with real-world problems, many researchers concentrate on approximation algorithms or heuristic approaches. Those approaches can find a near-optimal solution in a reasonable time. One of the most widely used approximation algorithms is Christofides algorithm [6]. Approximation algorithms run in polynomial time, but they often fail to find an optimal solution, and solution quality is not satisfactory. In practical applications, we always employ heuristic or local search algorithms. A basic heuristic algorithm is the 2-OPT technique. As a generalization, Lin-Kernighan (LKH) is one of the best heuristics for solving the symmetric TSP [7,8]. Meta-heuristic algorithms based on local search and swarm intelligence show good computational results of TSP solutions, and modified metaheuristic algorithms have also been proposed to solve many extensions of the TSP [9–11]. Besides, the Concorde TSP solver is a well-known solver that employs the cutting-plane-based approach [12]. Moreover, Google OR-Tools [13], combining local search algorithms and meta-heuristics can solve a large range of TSPs and vehicle routing problems. It is noted that machine learning-based approaches have received great success on solving the TSP. For instance, reinforcement learning was studied for algorithm selection [14]. Recently, many machine learning-based approaches have been proposed to solve various routing problems, where constraints and data of a problem are fixed and known before solving it [15–20].

The classic TSP supposes nodes and all costs are all known before solving. Actually, in real-world applications of logistics, the problem is dynamic, so the traditional static TSP model cannot model those logistics problems. Therefore, a number of dynamic models of the TSP were investigated. Dynamic customers and demands were introduced [10, 21–26]. As we discussed above, the problems from real-life applications usually have dynamic changes, and some constraints are stochastic. Machine learning-based approaches have a better ability to produce robust solutions in dynamic situations. However, there is few machine learning based approach to dealing with dynamic TSPs. In this paper, we study a traveling salesman problem with dynamic customers. We propose a sampling algorithm based on machine learning for the dynamic TSP with the case that the set of customers changes every day and allow online requests, *i.e.*, each day the set is different from other days, so we shall solve similar problems with the same road network but different customer sets. Different from existing dynamic TSP algorithms that are mainly based on metaheuristic algorithms, our method is a hybrid approach combining machine learning and sampling methods. It first employs machine learning techniques to train an MLP model by sampling tours of TSPs. To train model, we present an encoding method to transform travel states into feature vectors, and employ the current location and unvisited customer set to specify a state in a tour. After that, based on the trained model, we introduce how to construct a route to the problem and present an online sampling algorithm to deal with the situation that customers emerge during travel, where sampling of routes is performed based on the prediction by the trained model. We carried out extensive experiments to test our approaches on both randomly generated TSP instances and benchmarks from TSPLIB [27]. We analyzed comparative results of experiments intensively and showed that the proposed algorithm is able to produce much better solutions than the existing heuristics. Moreover, the results of runtimes also indicate that our approach is more efficient than other heuristic algorithms.

The remainder of this paper is organized as follows. The next section introduces notations and definitions used in the paper. Section 3 introduces our model based on MLP and the training method. In Section 4, we propose the prediction algorithm based on the trained model and the sampling algorithm for online dynamic TSP. Section 5 analyzes experimental results. Finally, we give some conclusions.

2. DYNAMIC TSP

In this section, we describe the TSP and dynamic TSP models we study in this paper. First, a TSP can be modeled by an undirected weighted graph $P = (V, E, W)$, where $V = \{1, 2, ..., n\}$ is a set nodes of the given graph, and E is a set of edges, and W is a weight matrix. We always use a positive integer to denote a node in V. Given a pair of nodes i and j, we use $(i, j) \in E$ to denote the edge whose two ends are i and j respectively, and $w(i,j) \in W$ is the weight of the edge (i, j). The graph P is a complete graph, where each pair of nodes is connected by an edge. We only consider symmetry situations in this paper, i.e., $w(i, j) = w(j, i)$. The weights can be distances of nodes, travel time, or travel cost. In this paper, we regard the weights as travel time.

In the classic TSP, the salesman starts from node 1 (depot), visits each other node in V exactly once, and returns back to the depot. A tour (solution) is a sequence of nodes in the order that the salesman visits. We use $S = <v_1, v_2, ..., v_n, v_1>$ to denote a tour, where $v_1 = 1$ and $v_i \in V\backslash\{1\}$ $(i = 2,3,...,n)$. The task of solving a TSP is to find a tour minimizing the sum of weights of all edges in the tour. Formally, the objective is defined by the following optimization problem:

$$\min\ cost(S)\ \ s.t.\ S \in SolSet \tag{1}$$

where $cost(S) = \sum_{(v_i,v_j)\in S} w(v_i, v_j)$ is the cost function, (v_i, v_j) is an ordered pair of adjacent nodes in the variable $S = <v_1, v_2, ..., v_n, v_1>$, and *SolSet* is the feasible solution set. We shall find a sequence of nodes for the variable S minimizing $cost(S)$.

Consider business in a retail logistics company, customers (nodes) may not be visited every day, so the customer set to serve is different in each day. We shall solve TSP instances with a subset of nodes repeatedly rather than the whole node set. Those TSP instances have the same road network data but node sets are different. Furthermore, customers' requests are sometimes online. They are unknown at the beginning of delivery in a day. In this context, we should re-plan the routing to serve as many requests as possible within a deadline. Based on the above discussion, we present dynamic TSP models. Two dynamic cases are described. The first one concerns dynamic customers, but the customers to be visited are known before solving it. We call this problem TSP with Dynamic Customers (TSPDC). We modify the description of classic TSPs and give the following definition.

Definition 1**.** Given a base TSP $P=(V, E, W)$ and a collection of node sets $C=\{V_1, V_2, ..., V_m\}$ such that $1 \in V_i$ ($V_i \subseteq V$, $1 \le i \le m$), a TSPDC case *w.r.t.* V_i is a subproblem of the base TSP P to find a tour S that starts from and ends at the node 1, and visits each other node in V_i exactly once.

All cases of the dynamic TSP compose a TSPDC. The task of solving the TSPDC is to solve all subproblems *w.r.t.* the node sets in C respectively and find the best solutions minimizing the sum of weights of all edges in the tours. Each set in the collection C can be regarded as a set of nodes that should be visited in each day. V_i may be different from others in C, so similar TSP instances should be solved repeatedly with the same road network but different node sets.

The second one is defined to deal with online requests. We only know a part of customers before starting delivery, and other customers emerge randomly, so the node set to be visited is divided into 2 subsets: V_o is revealed before service and V_d is revealed during the tour. When a node in V_d is revealed during tour, the solution should be recomputed to add the new node into the tour. We assume recomputation is only performed when the vehicle arrives at a node, and thus an online case is defined as follow.

Definition 2. An online TSPDC case is defined by a 5-tuple (P, v_c, T_c, V_r, T), where P is the base TSP, v_c is the current location (node), T_c is the current time, V_r is a remaining node set including unvisited nodes in V_o and new nodes in V_d, and T is a limitation time.

The task of solving an online case is to find a tour $S=<v_c, v_1,\dots,v_t,1>$ ($v_1,\dots,v_t \in V_r$) such that the vehicle arrives at the depot before T and maximizing the number of nodes in S. Suppose *SolSet* is the feasible solution set, the goal can be formally expressed as

$$\max|S| \;\; s.t.\; T_c + cost(S) \le T\,, \tag{2}$$

where $S \in SolSet$.

3. THE MODEL

3.1. The network structure

The MLP [28] is used and trained in our model. The structure of MLP is shown in Fig. 1. Though the structure of MLP is relatively simple and straightforward, it can achieve good training speed and good performance for dealing with the TSP. Our MLP consists of four layers: an input layer, an output layer, and two hidden layers. Since MLPs are fully connected, each node in a layer connects to every node in the following layer. The number of neurons in the input layer and output layer is the problem size, *i.e.* the number of total nodes in the given TSP instance. We employ the rectifier linear unit (ReLU) as the activation function for neurons in the hidden layers, and the softmax function in the output layer to map output values to the interval of (0,1). The cumulative sum of all output values after activation is 1.

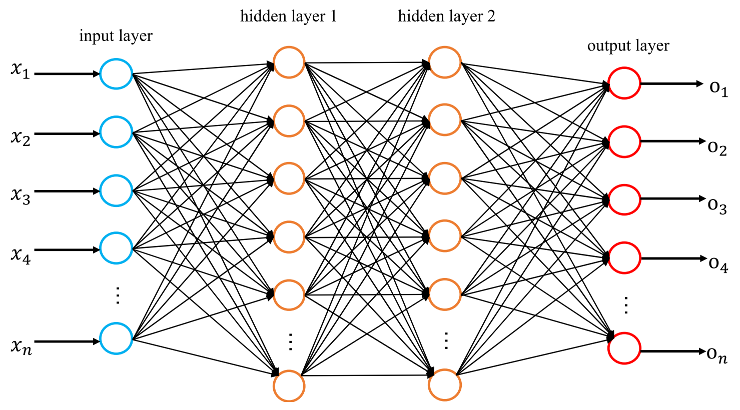

Fig. 1 – The structure of the MLP with an input layer, two hidden layers and an output layer.

In forward propagation, each neuron is linearly transformed as follow:

$$z_i^l = \sum_j w_{ij}^l x_j^{l-1} + b_i^l\,, \tag{3}$$

where w_{ij}^{l} is the weight from the *j*-th neuron in the $(l-1)$-th layer to the *i*-th neuron in the *l*-th layer, x_j^{l-1} is the input of the *j*-th neuron in the $(l-1)$-th layer, and b_i^l represents the bias of the *i*-th neuron in the *l*-th layer. After the linear transformation is computed, in the hidden layers, the ReLU function is applied to *z* for nonlinear transformation. ReLU activation function is computed as:

$$f(z_i) = \max(0, z_i)\,. \tag{4}$$

In the output layer, we use the softmax function to map output values to the interval (0,1), which represents the probability of each node selected. The softmax function is computed as:

$$f(z_i) = \exp(z_i) / \sum_k \exp(z_k)\,. \tag{5}$$

Loss function. We use the cross-entropy loss function to calculate loss of the model, which is often used for training classification problems. The loss function of one batch is calculated as:

$$L(X) = -1/B\sum_{i=1}^{B}\sum_{k=1}^{K} q(x_{ik})\log(p(x_{ik}))\,, \tag{6}$$

where $q(x_{ik})$ is the true probability, and $p(x_{ik})$ is the model predicted probability of the sample x_i belonging to the category *k*. *K* is the number of categories which equal to the problem size, and *B* is the batch size. According to the loss function *L*(*X*), we can calculate the update gradient of neural network parameter θ, where the update gradient of z_i is calculated as:

$$\nabla L(X) = \frac{\partial L(X)}{\partial z_i} = -1/B\sum_{i=1}^{B}\sum_{k=1}^{K} q(x_{ik})\frac{\partial log(p(x_{ik}))}{\partial z_i}\,. \tag{7}$$

According to $\nabla L(X)$, the network parameter θ is updated based on the mini-batch gradient descent method, and the parameter θ is updated as follow:

$$\theta \leftarrow \theta - \alpha\nabla L(X)\,, \tag{8}$$

where α is the parameter of update step. Suppose the number of nodes of a TSP instance is 20. The input vector *x* is transformed through two full connection layers to obtain the vector *z*, and then the vector *z* is input to the output layer. The selection probability *p* for each node is calculated through softmax activation.

3.2. Model training

Given a TSP $P = (V, E, W)$, we train the model based on supervised learning. We generate dynamic cases based on *P* and solve them by the state-of-the-art solver of the TSP (*i.e.*, Concorde TSP solver), and then convert the solutions (tours) into records in the training dataset. To generate a dynamic TSP instance, we choose a subset of nodes and solve the problem to obtain the best tour. Based on the selected nodes and the tour, we can produce a group of records. First, we choose a random integer s in the interval of $[n/2, n]$ as the number of nodes to be selected, where $n = |V|$ is the total number of nodes. Moreover, we select $s-1$ nodes randomly from $V\backslash\{1\}$. The selected nodes and the depot node 1 compose a node set, denoted as V'. After that, using Concorde TSP solver, we solve the TSP $P = (G, V, W)$ with the subset V' to obtain the best tour, denoted by *S*. We repeat the above steps to pick out 50000 dynamic cases and solve them. After obtaining the best tours, we convert those tours to a training dataset. For each tour *S*, we can produce $|S|-2$ records as follows. Pick up the first element v_c in *S*, and remove it from *S*. Thus, v_c is the current location, and *S* is the remaining node set to be visited. We then encode the pair (S, v_c) into a feature vector *x* with *n* length as input of our model, and the vector *y* as the label, which is a 0–1 vector with *n* length. The vector *x* is defined as $x = (x_1, x_2, ..., x_n)$, where $x_i = 0$ if the node *i* is not in $S\backslash\{1\}$ $(i \neq v_c,\ i = 1, ..., n)$; $x_i = 1$ if the node *i* is in $S\backslash\{1\}$ $(i = 1, ..., n)$; and $x_{v_c} = -1$. The vector $y = (y_1, ..., y_n)$ is defined as follows: $y_i = 1$, if *i* is the first element in *S*, $i = 1, ..., n$; $y_i = 0$, otherwise. Thus, a record (*x*, *y*) is produced. We repeat picking up the first element from *S* until $|S| < 2$, and thus $|S|-2$ records can be produced. We provide an example to indicate the encoding method. Suppose we have a subset V_0 and its tour *S*, where 1 is the depot and 2, 3, 4, 5, 6 are the customer nodes. Suppose the best tour is $S = <1, 4, 3, 2, 6, 5, 1>$, we can construct $|S|-2$ feature vectors shown in Table 1. Also, the labels are listed in the table.

Training process is performed as the following steps: (1) First, we shuffle the sample data randomly, and divide it into batches, where the batch size is 32; (2) For each batch, data is put into the model, and the loss is calculated according to the output values and labels; (3) With the loss value, the gradient descent method is used to update the parameters in the network; (4) Repeat steps (2)–(4) for 200 epochs to make the loss value converge, and the resultant parameters are saved. During training process, the dropout mechanism is employed to avoid overfitting.

Table 1

An example of feature vectors and labels with $S = <1, 4, 3, 2, 6, 5, 1>$

x	(-1,1,1,1,1,1)	(0,1,1,-1,1,1)	(0,1,-1,0,1,1)	(0,-1,0,0,1,1)	(0,0,0,0,1,-1)
y	(0,0,0,1,0,0)	(0,0,1,0,0,0)	(0,1,0,0,0,0)	(0,0,0,0,0,1)	(0,0,0,0,1,0)

4. ALGORITHMS FOR DYNAMIC TSP

In this section, we present algorithms to solve dynamic TSPs with the model we train. First, we discuss our prediction algorithm to construct a solution based on the MLP model, and then present our sampling algorithm to solve online dynamic TSPs.

4.1. The prediction algorithm for the TSPDC

We first discuss the prediction algorithm for the TSPDC. Based on the model, we present how to make decision and produce a solution to the TSPDC. Given $P = (V, E, W)$ and a subset V', the algorithm constructs a sequence of nodes by predicting each successor node iteratively. Decisions are made based on the trained model. In each step, it encodes the remaining unvisited nodes and the current location into an input vector of our model, and then it predicts the next node to be visited by the output vector. The next node will be assigned as the current location and added into the solution tour, so a solution is constructed by repeating prediction of successor visited nodes. Algorithm 1 indicates the procedure of solution construction.

Algorithm 1: The prediction algorithm for the TSPDC

```
    Input: A TSP $G = (V, E, W)$, a subset $V'$, and the trained model $M$
    Output: a solution $S$
1 $S \leftarrow \emptyset$; let $v_c \leftarrow 1$ be the depot, remove $v_c$ from $V'$ and append $v_c$ to $S$;
2 while $|V'| > 1$ do
3     $x \leftarrow encode(v_c, V')$; $mask \leftarrow maskvector(v_c, V')$;
4     $y \leftarrow M(x)$; // predict the next node
5     $v_{next} \leftarrow \text{argmax}(y \cdot mask)$;
6     remove $v_{next}$ from $V'$; append $v_{next}$ to $S$; $v_c \leftarrow v_{next}$;
7 append the element in $V'$ to $S$; append $v_1$ to $S$;
8 return $S$;
```

In Algorithm 1, the tour S is initialized by assigning the depot, and then it begins prediction. We define the function $encode(v_c, V')$ to calculate the input vector x. Similar with the previous section does, $x_i = 0$ if the node $i \notin V'$ except for $x_{v_c} = -1$, and $x_i = 1$ if $i \in V'$. The function $maskvector(v_c, V')$ returns a mask vector $mask \in \{0, 1\}^n$, where 1 means the existence in V' and 0 means the node is not included in V' (line 3). The function $M(x)$ returns the output vector of the model according to the input vector x (line 4). The output vector y is masked by the vector $mask$, and the node v with the maximum value of $y \cdot mask$ is picked out as the next one v_{next} to be visited (line 5). After the next node v_{next} to be visited is selected, it removes v_{next} from V' and appends v_{next} to the tour S, so the current location is changed to v_{next}. Prediction and selection of the next node will be repeated until there is only one node in V', and the node can be appended to the tour directly. Finally, the algorithm adds the depot as the last node in the tour.

4.2. The algorithm for online decision

Our algorithm for the online TSPDC constructs an initial solution by Algorithm 1, and then waits for new nodes and solves each online case to revise the current path repeatedly until the current time reaches the limitation. The online decision will be triggered to decide the next node when the vehicle arrives at a node, and update the next visited node if new requests emerged. Algorithm 2 indicates the main procedure of online decisions. Given a subset $V' = V_o \cup V_d$, where V_d is known at the beginning but V_o emerges randomly within T. It calculates an initial tour by Algorithm 1 with V_d as input of our trained model, and then initializes the current time T_c by 0 and $v_c = 1$. When the vehicle arrives at a node, it comes to a decision point, and the algorithm employs the current node v_c and the remaining node set V_r including all unvisited nodes currently known to compute the next node. If new nodes are added to V_r (line 6), the algorithm calls the function *sampling_solutions* to replan the path with nodes in V_r and obtains a new tour to update S (line 7). The function *sampling_solutions* will be introduced in Algorithm 3. The first node of S is taken as the next node (line 8). After the next node v_{next} is picked out, the algorithm moves the vehicle to v_{next} and updates the current time (lines 9–11). The algorithm terminates until $T_c \geq T$.

In Algorithm 3, the function *sample_solutions* samples solutions and finds the best tour as the resultant value. The algorithm samples solutions near the tour S calculated at the beginning, and selects the best solution as a new tour. We suppose the current location is the node v_c. Then, prediction is performed with the node set V_r' and the current location v_c (line 5). With a high probability (90%) it selects the successor of v_c in the original route where rand() returns a random number in [0,1], otherwise it takes the node predicted by the trained model as the next node. Then the current node is set to the next node to replan other remaining nodes so that a solution is constructed. After sampling H times (line 12), we get a set of solutions and pick out the best solution that can serve the most number of customers within the deadline T, and if there is more than one best solution, we choose the one with the smallest total travel time. In our experiment, H is set to 100.

Algorithm 2: The sampling algorithm for online TSPDC

```
Input: A dynamic TSP $P = (V, E, W)$ a subset $V'$ ($V_o \subseteq V'$, $V_d = V' \backslash V_o$) and deadline $T$
Output: null
1 let $S_{init}$ be the predicted tour by Algorithm 1;
2 $S \leftarrow S_{init}$; $V_r \leftarrow V_d$; $T_c \leftarrow 0$;
3 let $v_c$ be the depot in $V$; remove $v_c$ from $V_r$ and $S$;
4 while $T_c < T$ do
5     if $V_r$ is not an empty set then
6         if a new node in $V_o$ is added to $V_r$ after the last
          departure then
7             $S \leftarrow$ sample_solutions($S_{init}$, $V_r$, $v_c$, $T$);
8         let $v_{next}$ be the first node in $S$;
9         $v_c \leftarrow v_{next}$;
10        update $T_c$ as the time arriving at $v_{next}$;
11        remove $v_{next}$ from $V_r$ and $S$;
12    else
13        $v_c \leftarrow 1$, and move the vehicle to $v_c$;
14        update $T_c$ as the time arriving at $v_c$;
15        wait for a new customer;
```

Algorithm 3: sample_solutions(S, V_r, v_c, T)

```
Input: the original tour $S$, the remaining unvisited set $V_r$,
  the current location $v_c$ and time limitation $T$
Output: the new best tour $S_{best}$
1 $SolSet \leftarrow \emptyset$;
2 repeat
3     $S' \leftarrow \emptyset$; $V_r' \leftarrow V_r$; $v_c' \leftarrow v_c$;
4     while $V_r'$ is not empty do
5         let $v_{next}$ be the node predicted by Algorithm 1;
6         if rand() > 0.9 then
7             if successor of $v_c'$ in $V_r'$ then
8                 let $v_{next}$ be the successor of $v_c'$ in $S$;
9         append $v_{next}$ to $S'$, and remove $v_{next}$ from $V_r'$;
10        $v_c' \leftarrow v_{next}$;
11    add $S'$ to $SolSet$;
12 until sample H times;
13 select the tour $S_{best}$ that serves the most number of
   customers from $SolSet$ within $T$, tie breaking with the
   smallest travel time;
14 return $S_{best}$.
```

5. EXPERIMENTS

To evaluate the proposed model and algorithms for dynamic TSPs, we use a workstation with an Intel(R) Xeon(R) W-2145 CPU @ 3.70GHz and two RTX 2080 Ti GPUs, running Linux operating systems to test them. Our model and algorithms are implemented by Python 3.6.9 and Torch 1.7.1, and the model is trained based on Cuda 11.0.

We use two sets of benchmarks to test our model and algorithms. The first set is randomly generated instances, whose generation method was also used in testing existing machine learning-based approaches. We follow the method used in previous experiments to generate instances [15]. Locations of customers and the depot are randomly generated. For each customer and the depot, we generate a coordinate in the unit square [0, 1]×[0, 1] uniformly, and the weight of an edge is Euclidean distance. We set $n = 30$, 50, 100, respectively. For each value of n, we generate 5 TSP instances. The second benchmark is from TSPLIB, where 6 instances are chosen and tested: att48, berlin52, st70, pr76, gr96, and rd100. Those instances contain 48 to 100 nodes. We take 2-OPT, Christofides, LKH heuristic, and Google OR-tools to compare with our model. As we stated, LKH heuristic and Google Or-tools are efficient heuristics that can find a good solution in a short time. We use the Concorde TSP solver to find the best solutions for those benchmarks.

5.1. Comparative results of TSPDC

We first analyze convergence of training process. For each TSP instance, we train the model with 200 epochs. We select 4 instances ($n = 30$, 50, 100, respectively, and rd100) to show the training details and convergence. Figure 2 depicts the convergence curves of the 4 instances. From it, we can see that training almost converges after 40 epochs, and only a tiny improvement can be made on the loss value when the epoch is up to 200, so the training can be terminated. For the instance with $n = 30$, training process converges faster than others, and the convergence speed slightly declines with the size of instance growing.

Based on the trained model for each instance, we can predict solutions for the TSPDC by Algorithm 1. In the followings, we evaluate the prediction results for the TSPDC. For each instance, we select 1000 subsets for evaluating our approach, and also solve those subsets by the state-of-the-art algorithms, including heuristics and approximate algorithms. We select a subset of nodes randomly, whose size is a random integer in the interval $[n/2, n]$. The subset is the nodes to be visited, simulating daily requests of a retail logistics company. We compare the average travel time of tours produced by each algorithm and our model.

Table 2 indicates the results. The average travel time of 1000 dynamic cases for each algorithm is listed, and the solutions obtained by the Concorde TSP solver are listed as optimal results. The column "Model" is results of our approach. Gaps between our approach and the optimal solution are also listed. Moreover, results of instances from TSPLIB are included in the table. From it, we can see that our approach can give a very close result to the best solution, and it is far better than 2-OPT and Christofides. It is also easy to see that our approach can produce better results than the solutions of heuristics (*i.e.*, LKH and OR-tools) for each instance. The gaps between solutions of LKH and Google OR-tools and the optimal results are 1.88% and 2.42% on average, respectively. Those gaps are much bigger than ours. It is noted that most gaps of our approach do not exceed 1%, and our gap is 0.36% on average.

Table 2

Average travel time of 1000 dynamic cases

	Dataset	Method						
		2-OPT	Christofides	OR-Tools	LKH	Model	gap(%)	optimal
$n = 30$	S1	3.9726	4.1528	3.869	3.8867	3.8478	0.02%	3.847
	S2	4.1714	4.3192	3.9829	3.9696	3.9429	0.14%	3.9374
	S3	4.4455	4.6153	4.3002	4.4037	4.2857	0.03%	4.2846
	S4	3.7868	3.9888	3.6464	3.6816	3.6268	0.05%	3.6251
	S5	4.2586	4.4648	4.1265	4.1334	4.1008	0.04%	4.099
	mean	4.127	4.3082	3.985	4.015	3.9608	0.06%	3.9586
$n = 50$	S1	5.5711	5.6291	5.2008	5.169	5.1188	0.44%	5.0966
	S2	5.2509	5.2289	4.8918	4.904	4.829	0.21%	4.8191
	S3	5.5362	5.5725	5.219	5.1991	5.1076	0.35%	5.0898
	S4	5.1258	5.2464	4.8807	4.8047	4.7337	0.20%	4.7244
	S5	5.7437	5.7856	5.3309	5.3317	5.2558	0.22%	5.2444
	mean	5.4455	5.4925	5.1046	5.0817	5.009	0.28%	4.9949
$n = 100$	S1	8.3464	7.7313	7.1419	6.9489	6.9434	1.03%	6.8728
	S2	8.2329	7.4866	7.0546	6.8881	6.8633	0.80%	6.8086
	S3	7.815	7.2268	6.77	6.5635	6.5025	0.54%	6.4678
	S4	7.871	7.1583	6.7448	6.5895	6.4801	0.25%	6.4638
	S5	7.6995	7.099	6.5495	6.3972	6.3698	0.59%	6.3322
	mean	7.993	7.3404	6.8522	6.6774	6.6318	0.65%	6.589
TSPLIB	att48	30328	31772	29002	29675	28645	0.19%	28590
	berlin52	7157.6	7264.8	6755.1	6649.3	6566.7	0.32%	6546
	st70	664.2	647.6	602.3	599.7	588.3	0.35%	586.3
	pr76	107574	104575	97801	100316	95467	0.45%	95038
	gr96	526.8	482.4	455	451.4	445.2	0.92%	441.1
	rd100	8330	7686.3	7140	6961.7	6935.8	0.61%	6894

Furthermore, we draw box plots of the results to visualize summary statistics. We take TSP30-S1 as an example. For the instance, there are 1000 cases, and for each case, we calculate the best result among all algorithms and calculate the gaps between the best and the result of each algorithm. We draw box plots to show the distribution of those gaps. Figure 3 depicts the results, from which we can see that Model has the smallest gaps among heuristics, and the differences between our algorithm and others are significant. For a large part of cases, our method can produce the best solutions or high quantity solutions very close to the best obtained by the exact solver. Also, LKH can make near-optimal solutions, but the average is not as good as ours and the distribution interval is slightly larger than ours. It is obvious that OR-Tools is not comparative with our method, because the average result is worse and the distribution interval is larger compared with our method. It is also obvious that 2-OPT and Christofides got worse results than others.

Besides, we compare runtimes of those algorithms. Total runtimes of 1000 runs for each TSP instance are counted. Table 3 lists the comparative results in seconds. From it, we can see that it is the fastest one among all the comparative algorithms. It is about 11.0 times faster than LKH in total, and about 3.7 times than Concorde. The runtime of our approach grows linearly with the problem size increasing. Additionally, the runtime of constructing a solution by our approach is below 0.1s, and this runtime is short enough for practical applications.

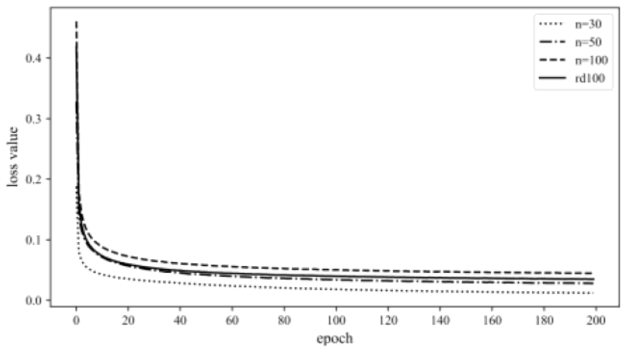

Fig. 2 – Convergence trend of loss function during model training, where $n = 30$, $n = 50$, $n = 100$ are randomly generated instances and rd100 is an instance with 100 nodes in TSPLIB.

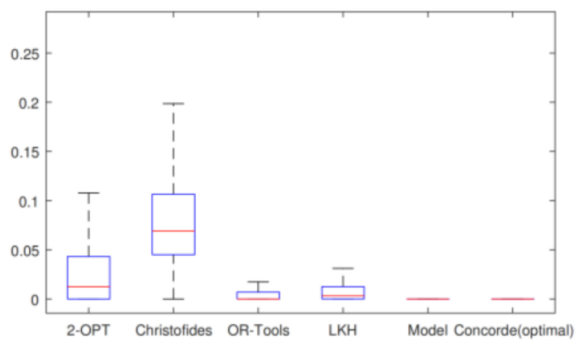

Fig. 3 – The box plot of 1000 cases in the instance TSP30-S1.

Table 3

Total runtimes of solving 1000 dynamic cases in seconds

Dataset	2-OPT	Christofides	OR-Tools	LKH	Model	Concorde
TSP-30	48	**5**	6	18	**5**	6
TSP-50	114	14	18	70	**11**	23
TSP-100	376	69	86	365	**23**	61
TSPLIB	1416	247	280	1294	**106**	591

5.2. Comparative results of online TSPDC

In this subsection, we test our sampling algorithm (Algorithm 2) for solving the online TSPDC, and show the computational results. We take the same instances as the experiment in Section 5.1 to test algorithms. To simulate the daily business, we generate 1000 cases randomly for each instance. The method used here is similar with the generation of the TSPDC. We first select the number of nodes s varying from $n/2$ to n, where n is the total number of nodes in the instance, and then choose s customers from the node set, denoted as V'. After that, we divide V' into two sets V_d and V_o. V_d is the node set in which nodes are known at the beginning and nodes in V_o emerge after it starts. In the following experiment, the nodes from V' are put into V_o with the proportion of p. We also generate an emerging time for each customer in V_o, denoted by t, and t is a random number from interval $[0, T]$, where T is the complete time of the best solution obtained by Concorde.

We compare our sampling approach (denoted by Model-Sampling) with LKH, since LKH is the best one among heuristic algorithms. It is noted that LKH cannot solve the online TSP problems directly. Therefore, we modify it to be able to solve the online version. LKH is triggered to re-plan the path if there is a new customer emerged. Moreover, OR-Tools is modified using the same approach as LKH. Christofides, 2-OPT and OR-Tools are not considered in solving online cases because they perform worse than LKH. We also compare the sampling approach with a greedy approach based on the trained model (denoted by Model-Greedy), where at each node it replans the tour by predicting the action directly. Besides, we compute the posterior solution and list the average result. We suppose all nodes in V_o are known at the beginning and solve the problem by Concorde. It is clear that a solution to the online case cannot be better than the best solution to the case knowing all nodes at the beginning. The posterior solution is used as time limitation T to evaluate results produced by our sampling approach and calculate the proportion of customers that can be served as well as the average travel time within the limitation.

Table 4 gives the average results when $p = 20\%$. We list the ratio of served customers to total customers and the average travel time of each instance over 1000 cases. It is clear that our sampling algorithm is better than Model-Greedy, since more customers can be served within the limitation. It is also noted that Model-Sampling performs much better than LKH and OR-Tools, though the travel time of Model-Sampling is slightly larger than the two heuristic algorithms. Within the limitation of travel time, Model-Sampling has better results on both random instances and TSPLIB benchmarks. In addition, the average runtime of our approach is within 3s for all cases, and each decision costs less than 1s though it requires more solving time than LKH and Model-Greedy in total. Therefore, our approach is effective for practical purposes.

Table 4

Average completion rate on solving online TSPDC cases ($p = 20\%$)

		LKH		Model-Greedy		Model-Sample		
	Dataset	ratio	travel time	ratio	travel time	ratio	travel time	limitation
$n = 30$	S1	0.85	3.3992	0.81	3.3366	**0.89**	3.3595	3.562
	S2	0.9	3.4575	0.87	3.4552	**0.91**	3.4376	3.665
	S3	0.83	3.8069	0.8	3.8038	**0.88**	3.8163	4.0134
	S4	0.75	3.1281	0.8	3.1473	**0.88**	3.1634	3.333
	S5	0.8	3.6671	0.79	3.6313	**0.87**	3.679	3.8294
	mean	0.83	3.4918	0.81	3.4749	**0.89**	3.4912	3.6806
$n = 50$	S1	0.79	4.5152	0.81	4.5236	**0.86**	4.5737	4.6831
	S2	0.85	4.2761	0.77	4.2077	**0.86**	4.272	4.4075
	S3	0.77	4.4282	0.74	4.4242	**0.85**	4.4948	4.6154
	S4	0.82	4.231	0.76	4.1803	**0.86**	4.2288	4.3563
	S5	0.8	4.6762	0.79	4.6577	**0.86**	4.707	4.8364
	mean	0.8	4.4254	0.78	4.3987	**0.86**	4.4553	4.5798
$n = 100$	S1	0.75	6.088	0.72	6.0814	**0.82**	6.1585	6.235
	S2	0.76	6.0423	0.67	5.9411	**0.8**	6.0883	6.1597
	S3	0.74	5.8105	0.68	5.7485	**0.82**	5.8571	5.9363
	S4	0.72	5.735	0.64	5.6435	**0.81**	5.7952	5.876
	S5	0.74	5.6907	0.65	5.6238	**0.8**	5.7567	5.8293
	mean	0.74	5.8733	0.67	5.8076	**0.81**	5.9312	6.0073
TSPLIB	att48	0.76	25089.7	0.71	24820.9	**0.84**	25502.3	26189.3
	berlin52	0.81	5741.5	0.77	5768.4	**0.85**	5837.6	5973.4
	st70	0.75	510	0.68	504.4	**0.82**	517.6	525.2
	pr76	0.77	84074.8	0.75	83504.1	**0.84**	84652.8	86634.9
	gr96	0.79	391.7	0.68	387.8	**0.81**	395.5	401.7
	rd100	0.72	6076.2	0.62	6015.7	**0.8**	6156.8	6234.6

6. CONCLUSION

The traveling salesman problem is a fundamental combinatorial optimization problem. It is an important tool for modeling various routing problems. In this paper, we investigate a dynamic TSP in a retail logistics company. It delivers or picks up cargoes of customers, and the road network is static in a period, but the node set to be visited in a day is different from other days. We proposed a machine learning-based method to solve the dynamic TSP. Moreover, we study the online version of the dynamic TSP, where a part of customers are allowed to submit their requests during service. We propose a sampling algorithm on the base of the trained model, to find a solution that can visit as many customers as possible in a limited time. We test our approaches on both randomly generated TSP instances and benchmarks from TSPLIB, and compare our approaches with existing well-known heuristic algorithms. The results demonstrate our approach has a good ability to solve the dynamic TSP and produce high-quality solutions. Also, the results show that the efficiency of our approach is comparable with highly optimized TSP solvers.

ACKNOWLEDGEMENTS

We would like to thank anonymous reviewers for useful suggestions. This work was supported by the National Natural Science Foundation of China (No. 61972063).

REFERENCES

1. M. BELLMORE, G.L. NEMHAUSER, *The traveling salesman problem: A survey*, Operations Research, **16,** *3,* pp. 538–558, 1968.
2. C.E. DODU, M. ANCĂU, *An efficient algorithm for lot permutation flow shop scheduling problem*, Proceedings of the Romanian Academy, Series A, **22**, *3*, pp. 231–238, 2021.
3. J. GAO, R. CHEN, W. DENG, *An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem*, International Journal of Production Research, **51**, *3*, pp. 641–651, 2013.
4. R.M. KARP, *Reducibility among combinatorial problems,* In: Complexity of Computer Computations (eds. R.E. Miller, J.W. Thatcher, J.D. Bohlinger), pp. 85–103, Springer, 1972.
5. F. FOCACCI, A. LODI, M. MILANO, D. VIGO, *Solving TSP through the integration of or and CP techniques,* Electronic Notes in Discrete Mathematics, **1**, pp. 13–25, 1999.
6. N. CHRISTOFIDES, *Worst-case analysis of a new heuristic for the travelling salesman problem,* Technical Report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.
7. S. LIN, B.W. KERNIGHAN, *An effective heuristic algorithm for the traveling-salesman problem*, Operations Research, **21**, *2*, pp. 498–516, 1973.
8. D. KARAPETYAN, G. GUTIN, *Lin–kernighan heuristic adaptations for the generalized traveling salesman problem,* European Journal of Operational Research, **208**, *3*, pp. 221–232, 2011.
9. M. YOUSEFIKHOSHBAKHT, *Solving the traveling salesman problem: A modified metaheuristic algorithm*, Complexity, Art. 6668345, 2021.
10. P. STODOLA, K. MICHENKA, J. NOHEL, M. RYBANSKY, *Hybrid algorithm based on ant colony optimization and simulated annealing applied to the dynamic traveling salesman problem,* Entropy, **22**, *8*, Art. 884, 2020.
11. E. BARTOLINI, D. GOEKE, M. SCHNEIDER, M. YE, *The robust traveling salesman problem with time windows under knapsack-constrained travel time uncertainty,* Transportation Science, **55**, *2*, pp. 371–394, 2021.
12. D. APPLEGATE, R. BIXBY, V. CHVÁTAL, W. COOK, *TSP cuts which do not conform to the template paradigm,* in: *Computational Combinatorial Optimization* (eds. M. Jünger, D. Naddef), Springer, 2001, pp. 261–303.
13. L. PERRON, *Operations research and constraint programming at Google,* in: *International Conference on Principles and Practice of Constraint Programming* (ed. J. Lee), p. 2, Lecture Notes in Computer Science, Vol. 6876, Springer, 2011.
14. I. BELLO, H. PHAM, Q.V. LE, M. NOROUZI, S. BENGIO, *Neural combinatorial optimization with reinforcement learning,* 5th International Conference on Learning Representations, 2017.
15. M. NAZARI, A. OROOJLOOY, L.V. SNYDER, M. TAKÁC, *Reinforcement learning for solving the vehicle routing problem,* Annual Conference on Neural Information Processing Systems 2018, pp. 9861–9871, 2018.
16. J.J.Q. YU , W. YU, J. GU, *Online vehicle routing with neural combinatorial optimization and deep reinforcement learning,* IEEE Transactions on Intelligent Transportation Systems, **20**, *10*, pp. 3806–3817, 2019.
17. Y. HU, Y. YAO, W.S. LEE, *A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs,* Knowledge-Based Systems, **204**, art. 106244, 2020.
18. M.O.R. Prates, P.H.C. Avelar, H. Lemos, L.C. Lamb, M. Vardi, *Learning to solve NP-complete problems: A graph neural network for decision TSP,* The Thirty-Third AAAI Conference on Artificial Intelligence, pp. 4731–4738, AAAI Press, 2019.
19. P.R. de Costa, J. Rhuggenaath, Y. Zhang, A. Akcay, *Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning,* Proceedings of The 12th Asian Conference on Machine Learning, **129**, pp. 465–480, 2020.
20. Y. HU, Z. ZHANG, Y. YAO, X. HUYAN, X. ZHOU, W.S. LEE, *A bidirectional graph neural network for traveling salesman problems on arbitrary symmetric graphs,* Engineering Applications of Artificial Intelligence, **97**, Art. 104061, 2021.
21. A. GUPTA, V. NAGARAJAN, R. RAVI, *Approximation algorithms for VRP with stochastic demands,* Operations Research, **60**, *1*, pp. 123–127, 2012.
22. R. TADEI, G. PERBOLI, F. PERFETTI, *The multi-path traveling salesman problem with stochastic travel costs,* EURO Journal on Transportation and Logistics, **6**, *1*, pp. 3–23, 2017.
23. A. TORIELLO, W.B. HASKELL, M. POREMBA, *A dynamic traveling salesman problem with stochastic arc costs,* Operations Research, **62**, 5, pp. 1107–1125, 2014.
24. C. ARCHETTI, D. FEILLET, A. MOR, M.G. SPERANZA, *Dynamic traveling salesman problem with stochastic release dates,* European Journal of Operational Research, **280**, *3*, pp. 832–844, 2020.
25. S. CHEN, R. CHEN, G.-G. WANG, J. GAO, A.K. SANGAIAH, *An adaptive large neighborhood search heuristic for dynamic vehicle routing problems,* Computers & Electrical Engineering, **67**, pp. 596–607, 2018.
26. S. CHEN, R. CHEN, J. GAO, *A modified harmony search algorithm for solving the dynamic vehicle routing problem with time windows,* Scientific Programming, Art. 1021432, 2017.
27. G. REINELT, *TSPLIB – a traveling salesman problem library,* ORSA Journal on Computing, **3**, *4*, pp. 376–384, 1991.
28. F. MURTAGH, *Multilayer perceptrons for classification and regression,* Neurocomputing, **2**, *5*, pp. 183–197, 1991.

Received November 10, 2021