Near-Optimal Traveling Salesman Solution with Deep Attention

Natdanai Kafakthong, Krung Sinapiromsaran
Department of Mathematics and Computer Science, Chulalongkorn University, Bangkok, Thailand, 10330

***Abstract*—The Traveling Salesman Problem (TSP) is a well-known problem in computer science that requires finding the shortest possible route that visits every city exactly once. TSP has broad applications in logistics, routing, and supply chain management, where finding optimal or near-optimal solutions efficiently can lead to substantial cost and time reductions. However, traditional solvers rely on iterative processes that can be computationally expensive and time-consuming for large-scale instances. This research proposes a novel deep learning architecture designed to predict optimal or near-optimal TSP tours directly from the problem's distance matrix, eliminating the need for extensive iterations to save total solving time. The proposed model leverages the attention mechanism to effectively focus on the most relevant parts of the network, ensuring accurate and efficient tour predictions. It has been tested on the TSPLIB benchmark dataset and observed significant improvements in both solution quality and computational speed compared to traditional solvers such as Gurobi and Genetic Algorithm. This method presents a scalable and efficient solution for large-scale TSP instances, making it a promising approach for real-world traveling salesman applications.**

***Keywords*—Traveling salesman problem; deep learning; genetic algorithm**

I. Introduction

The Traveling Salesman Problem (TSP) [1] is a classic combinatorial optimization problem that involves finding the shortest possible tour for a salesman to visit a given set of cities, exactly once. Each city is connected by a set of weighted edges that represent the distances between cities, and the objective is to minimize the total travel distance. Despite its seemingly simple formulation, TSP is known to be NP-hard, meaning the computational complexity grows exponentially with the number of cities, making it extremely difficult to solve for large instances within a reasonable time frame.

TSP has significant importance in both theoretical research and practical applications. It serves as a benchmark for optimization techniques and has a wide range of real-world uses, such as logistics, manufacturing, and routing problems. Solving TSP efficiently can result in substantial cost savings and improved resource management in industries that rely on optimized routing and scheduling. Beyond its practical implications, TSP also represents a fundamental challenge in computational theory, driving the development of new algorithms and techniques that have broader applications in other complex optimization problems.

Several methods have been developed to solve TSP [1], including exact algorithms and heuristics. Exact methods, such as brute force, branch-and-bound, and dynamic programming, attempt to find the optimal solution but they are computationally expensive and infeasible for large-scale problems due to their exponential time complexity. On the other hand, heuristic and metaheuristic approaches like genetic algorithms (GA) [5], simulated annealing, and ant colony optimization [9] offer approximate solutions by exploring the solution space more efficiently without guaranteeing an optimal result. These methods are often favored for large instances due to their ability to provide good solutions within a reasonable time frame. The effectiveness of these solvers is therefore highly dependent on the initial solution or tour. A poor starting solution can lead to long convergence times and suboptimal solutions, while a good initial solution can significantly reduce the number of iterations and improve the overall performance of the solver.

A near-optimal initial solution can improve the performance of traditional solvers by reducing the search space and accelerating convergence. Deep learning models, trained on problem data such as city distances, can predict a near-optimal tour. However, these models may not always predict a valid tour that meets the tour constraints. In such cases, the predicted solution must be refined or corrected before it is passed to a traditional solver.

This paper proposes a hybrid approach that uses deep learning to predict an initial solution for TSP and introduces an algorithm to reformulate this prediction into a valid tour. If the predicted solution from the deep learning model is not a valid tour, the tour correction algorithm adjusts it by ensuring that each city is visited exactly once and the path forms a valid loop. This refined initial solution is then fed into traditional optimization methods, such as GA or Gurobi [23], significantly reducing the time and iterations required to reach an optimal or near-optimal solution. The key advantage of this approach is that even if the predicted tour is not optimal tour, the generated tour is not differ from the optimal tour too much.

This hybrid method offers a balance between speed and accuracy. By leveraging deep learning to predict a strong initial solution and correcting it as needed, the traditional solvers can focus on fine-tuning, which reduces computational time and allows for efficient optimization of large TSP instances.

A. Contributions

- This paper introduces an approach to reduce the computational time of traditional TSP solvers by providing a near-optimal tour as the starting solution, significantly improving solver solution time.
- The proposed deep learning architecture, TSPNet, utilizes an attention mechanism in the architecture to handle combinatorial tasks in TSP.

- An algorithm is presented to convert the predicted solution or tour from TSPNet, which may not always be a valid tour, into a proper initial tour that satisfies the TSP constraints.
- The effectiveness of TSPNet, combined with traditional solvers, is evaluated on the TSPLIB dataset [25], demonstrating substantial reductions in computational time.

The remainder of this paper is organized as follows. Section II reviews related work on TSP solvers and deep learning approaches. Section III introduces the mathematical notation for the TSP. Section IV discusses traditional TSP solvers and the use of deep learning models for predicting initial tours. Section V details the methodology, including synthetic dataset generation, TSPNet architecture, input preprocessing, training, and the solving process. Section VI presents experimental results on the TSPLIB dataset and evaluates the integration of TSPNet with a Genetic Algorithm. Section VII discusses findings, limitations, and future directions, and Section VIII concludes the paper.

II. Related Work

The Traveling Salesman Problem [1] is a well-known NP-hard combinatorial optimization problem that seeks to determine the shortest possible route that visits a set of cities exactly once and returns to the origin city. Recent advancements in deep learning and reinforcement learning have provided innovative approaches to tackle this complex problem. This section will explore various methodologies, including neural networks, genetic algorithms, and hybrid models that integrate these techniques.

One significant approach to solving TSP involves the use of deep reinforcement learning (DRL). DRL formalizes TSP as a sequential decision-making problem, allowing an agent to select the next city to visit at each step. This method leverages the powerful generalization capabilities of deep neural networks, yielding impressive results in solving TSP instances [2], [3]. For instance, Bello et al. demonstrated the effectiveness of a pointer network trained via reinforcement learning, which significantly improved performance on TSP tasks [2]. Additionally, the incorporation of evolutionary algorithms with DRL has shown promise in multi-objective optimization scenarios, further enhancing the solution quality for TSP variants [3].

Another noteworthy methodology is the application of Hopfield neural networks, which have been historically significant in solving combinatorial optimization problems, including TSP. The Hopfield network utilizes an energy function to find optimal tours by minimizing the total travel distance [4]. This approach has been validated through various studies, demonstrating its capability to outperform traditional computational methods in specific instances of the TSP [4]. Moreover, the integration of chaotic neural networks has been proposed to enhance the performance of Hopfield networks, providing a novel perspective on TSP solutions [4].

Genetic algorithms (GAs) also play a crucial role in addressing the TSP [5]. These algorithms mimic the process of natural selection to iteratively improve solutions. Recent studies have highlighted the effectiveness of hybrid genetic algorithms that combine traditional GA techniques with neural networks, leading to improved convergence rates and solution quality for TSP [6], [7]. Furthermore, the application of multi-colony ant systems and particle swarm optimization has been explored as alternative heuristic methods for solving TSP, showcasing the versatility of approaches available to researchers [8], [9].

Recent studies have highlighted the effectiveness of deep reinforcement learning (DRL) in developing heuristics for TSP. For instance, Kool et al. introduced a method called Deep Policy Dynamic Programming, which integrates machine learning with dynamic programming to yield near-optimal solutions for TSPs with up to 100 nodes, demonstrating competitive performance against established solvers like LKH [10]. Similarly, Perera et al. utilized pointer networks within a multi-objective deep reinforcement learning framework, which enabled the model to generalize from smaller TSP instances to larger ones, achieving optimal solutions for TSPs with over 1000 cities [11]. These advancements illustrate the capacity of deep learning models to learn and adapt heuristics that can significantly improve solution quality and computational efficiency.

Moreover, the integration of graph neural networks (GNNs) has further enhanced the ability to solve TSP. GNNs can effectively represent the problem as a graph, allowing for the exploration of various routes and the optimization of path selection through learned embeddings. This approach has been shown to outperform traditional methods by providing a more structured understanding of the problem space [12]. The ability of deep learning models to learn from data and improve their performance over time is particularly beneficial in combinatorial optimization, where the search space is vast and complex [13].

In conclusion, the integration of deep learning, reinforcement learning, and genetic algorithms has significantly advanced the methodologies available for solving the Traveling Salesman Problem. These approaches not only enhance the efficiency of finding optimal solutions but also expand the applicability of TSP solutions to various real-world scenarios, such as logistics and route planning. The ongoing research in this field continues to evolve, promising even more sophisticated techniques for tackling this enduring challenge in combinatorial optimization.

III. Mathematical Notation for the Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is an optimization problem where a set of n cities must be visited exactly once, and the goal is to find the shortest possible route that visits each city and returns to the starting point. Mathematically, the TSP can be modeled as an integer linear programming (ILP) problem. To represent this mathematically, let $G = (V, E)$ be a complete graph where V is the set of cities, $|V| = n$, and E is the set of edges. Each edge $(i, j) \in E$ is associated with a distance or cost d_{ij}. Define the binary decision variable x_{ij} as follows,

$$x_{ij} = \begin{cases} 1 & \text{if the tour visits city } i \text{ and then city } j, \\ 0 & \text{otherwise.} \end{cases}$$

The objective is to minimize the total travel cost or total tour length, which can be written as,

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} d_{ij} x_{ij}$$

subject to the following constraints.

1) Visit each city exactly once,

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1 \quad \forall i = 1, 2, \ldots, n.$$

This ensures that each city i is left exactly once.

2) Enter each city exactly once,

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \quad \forall j = 1, 2, \ldots, n.$$

This ensures that each city j is entered exactly once.

3) Subtour elimination, to prevent smaller loops that don't include all cities (sub tours), the following constraint is applied for all subsets $S \subset V$, where $|S| \geq 2$,

$$\sum_{i\in S} \sum_{j\in S, j\neq i} x_{ij} \leq |S| - 1.$$

This constraint ensures that the solution forms a single tour that includes all cities.

The optimal solution to TSP is the assignment of x_{ij} values (1 or 0) that minimize the total tour length while satisfying the constraints that ensure a valid tour. However, finding this optimal tour is computationally challenging due to the combinatorial nature of the problem. As the number of cities increases, the number of possible tours grows factorially, making it impractical for an exact algorithm to solve large instances within reasonable time limits.

In the context of solving TSP, a distance matrix plays a fundamental role. The distance matrix $D = [d_{ij}]$ is an $n \times n$ matrix, where d_{ij} represents the distance or cost of traveling from city i to city j. Mathematically, it is defined as,

$$D = \begin{bmatrix} 0 & d_{12} & d_{13} & \ldots & d_{1n} \\ d_{21} & 0 & d_{23} & \ldots & d_{2n} \\ d_{31} & d_{32} & 0 & \ldots & d_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & d_{n3} & \ldots & 0 \end{bmatrix}. \tag{1}$$

The $d_{ii} = 0$ since there is no distance associated with staying in the same city. This matrix forms the core input for solving TSP, as it provides the necessary information about the travel costs between every pair of cities.

IV. TSP Solvers

Traditional solvers for the Traveling Salesman Problem (TSP) rely heavily on the distance matrix [15], [16], [17], a fundamental representation that captures the pairwise distances between cities. This matrix plays a central role in various exact and heuristic algorithms, such as branch-and-bound [18], [19], dynamic programming [20], and cutting-plane methods [21], where it helps determine the shortest path between cities. For example, in branch-and-bound algorithms, the distance matrix is used to calculate lower bounds and prune non-optimal tours efficiently. Similarly, heuristic algorithms like the nearest neighbor or genetic algorithms use the matrix to guide their search for near-optimal solutions by iteratively considering the shortest available edges between cities.

The distance matrix provides essential information for constructing and refining the solution to TSP, making it indispensable for solving this combinatorial problem. Solvers like Gurobi [23], Concorde [24], and others leverage the distance matrix at each step of their optimization process, ensuring the best possible path is found given the constraints.

In line with these solvers, our proposed deep learning model also takes the distance matrix as an input. Instead of manually optimizing over possible routes, our model learns to predict the optimal tour from the structure of the distance matrix itself. By incorporating attention mechanisms, the model can effectively learn which city connections contribute most to form the optimal solution. The use of a distance matrix thus remains central, both in traditional algorithms and in modern machine learning approaches for solving TSP.

A. Predicting the Optimal Tour Using a Deep Learning Model

The complexity of solving the traveling salesman problem arises from the factorial growth in the number of possible tours as the number of cities increases, making it computationally challenging for exact algorithms to handle a large-scale instance. As previously discussed, deep learning has emerged as a promising approach for approximating optimal solutions of TSP, offering a more efficient alternative to traditional methods.

One innovative approach leverages deep learning models to predict optimal sub-tours or the entire tour based on historical data or synthetically generated datasets. In this framework, TSP can be formulated by using a distance matrix $D = [d_{ij}]$, which encodes the pairwise distances between cities. The deep learning model is trained to output a matrix of probabilities $P = [p_{ij}]$, where each element $p_{ij} \in [0, 1]$ represents the likelihood of a connection between city i and city j appearing in the optimal tour. This probability matrix serves as the predicted adjacency matrix for TSP, with the highest probability in each row corresponding to the city that is most likely connected to city i in the optimal tour.

The model, denoted by f_θ with parameters θ, processes the distance matrix D and outputs the adjacency matrix P,

$$f_\theta(D) = P = \begin{bmatrix} p_{11} & p_{12} & \ldots & p_{1n} \\ p_{21} & p_{22} & \ldots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \ldots & p_{nn} \end{bmatrix}. \tag{2}$$

Each row of P represents a probability distribution over all cities for the next destination from city i. To obtain the predicted tour, the argmax function is applied to each row, selecting the city with the highest probability, resulting in the vector T of optimal adjacent cities,

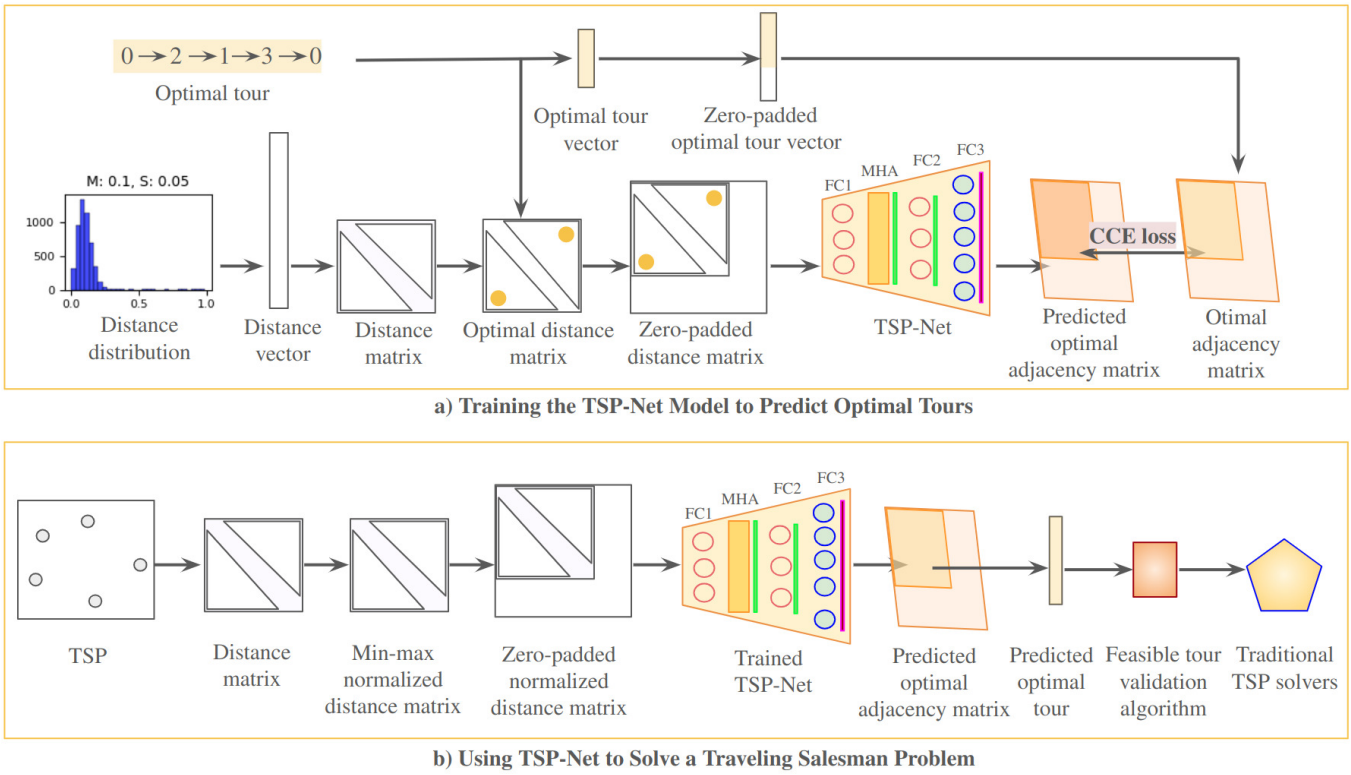


Fig. 1. Overview of the TSPNet framework: (a) Training involves generating distance matrices from a log-normal distribution and corresponding optimal tours to train the TSP model. (b) The trained model predicts a feasible tour, which is validated and can serve as an initial solution for traditional solvers.

$$T = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix} = \begin{bmatrix} \text{argmax}\{p_{11}, p_{12}, \ldots, p_{1n}\} \\ \text{argmax}\{p_{21}, p_{22}, \ldots, p_{2n}\} \\ \vdots \\ \text{argmax}\{p_{n1}, p_{n2}, \ldots, p_{nn}\} \end{bmatrix}. \quad (3)$$

In this representation, T contains the indices of cities selected as the next steps in the predicted tour. The predicted tour can then be reconstructed from T, forming a solution to TSP. This approach allows the deep learning model to approximate optimal solutions by learning patterns in city connections from the distance matrix efficiently, even for large-scale instances.

V. Methodology

This paper introduces TSPNet, a deep learning model designed to predict the optimal tour for the traveling salesman problem by learning the relationships between cities from distance matrices. The methodology involves two key stages: first, the training process of TSPNet, which includes generating synthetic training data, constructing input instances, and defining optimal tour labels for the model. The architecture of TSPNet is also detailed in a subsequent section. Second, TSPNet's predictions are extended to refine the solution using traditional solvers like genetic algorithms. By using TSPNet's predicted tour as an initial solution, these solvers require fewer iterations to reach an optimal or near-optimal solution. The next subsection discusses the dataset generation process for training TSPNet.

A. Synthetic TSP Training Dataset

TSP necessitates the generation of a distance matrix that accurately represents the distances between cities. In this study, a log-normal distribution is employed due to its capacity to model positively skewed data, which is often observed in real-world distance measurements.

A random variable X is said to be log-normally distributed if its natural logarithm, $Y = \ln(X)$, follows a normal distribution, that is

$$Y \sim \mathcal{N}(\mu, \sigma^2)$$

where μ is the mean and σ is the standard deviation of the underlying normal distribution. Consequently, X can be expressed as

$$X = e^Y \implies X \sim \text{Log-Normal}(\mu, \sigma^2).$$

This property ensures that the generated distances D are strictly positive, satisfying the non-negativity constraint required for distance metrics. A list of the distributions used to generate the training distances is shown in Fig. 2.

The figure presents the parameters used to generate various cases of log-normal distributions. This diversity allows the model to encounter different patterns of distances, enhancing its generalization ability and efficiency. Normally, input instances of deep learning will be normalized by min-max normalization before passing into the deep learning model. They are scaled into $[0, 1]$ then the training data of TSPNet will focus on generating training data in the range $[0, 1]$.

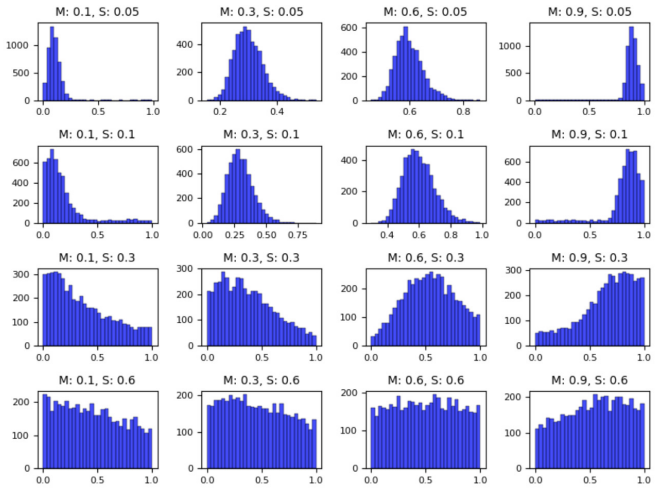


Fig. 2. Log-normal distributions for generating training distances.

For generating an input TSP instance of n cities or nodes from the log-normal distribution, the synthetic TSP datasets are executed as the following steps. The function described can be expressed mathematically as follows:

- n is the number of cities.
- μ is the mean of the generated distances.
- σ is the standard deviation of the generated distances.
- s is the scaling factor.
- λ is the shape parameter of the log-normal distribution.
- K is the number of unique tours to generate.
- $\mathcal{S}$ is the set of all cities $\{0, 1, 2, \ldots, n-1\}$.
- t_h is the h-th optimal tour T.

Step 1. Generate the distances: The generated distances are sampled from a log-normal distribution with parameters $\log(s)$ and λ,

$$d_{ij} \sim \text{LogNormal}(\log(s), \lambda), \forall i, j \in \{1, \ldots, \frac{n(n-1)}{2}\}, i < j.$$

Step 2. Adjust the mean and the standard deviation: Normalize the generated distances to have mean μ and standard deviation σ,

$$d'_{ij} = \frac{d_{ij} - \bar{d}}{\text{std}(d)} \cdot \sigma + \mu$$

where $\bar{d}$ is the mean of d_{ij} and $\text{std}(d)$ is the standard deviation. Step 3. Replace out-of-bound values: If $d'_{ij} < 0$ or $d'_{ij} > 1$, replace it with a uniformly random value within bounds,

$$d'_{ij} = \text{Uniform}(0, 1), \quad \text{if } d'_{ij} \notin [0, 1].$$

Step 4. Construct the distance matrix: The distance matrix D is symmetric and filled using the adjusted distances,

$$D_{ij} = D_{ji} = d'_{ij}, \quad \forall i < j.$$

The diagonal elements are set to zero

$$D_{ii} = 0, \quad \forall i.$$

Step 5. Create an optimal tour vector T: Defining a permutation t_h of the city indices,

$$t_h : \mathcal{S} \to \mathcal{S}, \quad \text{for } h = 1, \ldots, K.$$

Each t_h is a random shuffle of the set $\mathcal{S}$.

Step 6. The distance values in the distance matrix D_h corresponding to the cities in the optimal tour vector t_h will be reduced to ensure the distance matrix reflects the optimal tour.

Step 7. Convert the optimal tour vector t_h to be adjacency matrix using the one-hot-encoding technique for training the TSPNet model.

Thus, the distance matrix $D \in \mathbb{R}^{n \times n}$ is constructed based on the adjusted and bounded distances. By employing the log-normal distribution in this manner, the resulting distance matrix is both realistic and statistically appropriate, facilitating the effective training of the TSPNet model to predict the optimal adjacency matrix for varying configurations of cities.

B. TSPNet Input Preprocessing

To prepare the data for training the TSPNet model (as illustrated in Fig. 1(a), both the distance matrices and the corresponding optimal tour vectors are first created. As the input size for deep learning models is generally fixed, this research limits the maximum number of cities N in TSP to 200. To accommodate TSP instances with fewer than 200 cities, the distance matrices and optimal tour vectors are padded with zeros.

Let D be a distance matrix of size $n \times n$, where $n \leq N$, represents a TSP with n cities. The distance matrix D for n cities is structured as follows:

$$D = \begin{bmatrix} 0 & d_{12} & d_{13} & \ldots & d_{1n} \\ d_{21} & 0 & d_{23} & \ldots & d_{2n} \\ d_{31} & d_{32} & 0 & \ldots & d_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & d_{n3} & \ldots & 0 \end{bmatrix}. \tag{4}$$

For cases where $n < N$, D is padded to match the maximum size of $N \times N$, resulting in the following matrix,

$$D' = \begin{bmatrix} D & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{5}$$

Similarly, the optimal tour vector T of size $n \times 1$ is padded to size $N \times 1$ as follows:

$$T' = \begin{bmatrix} T \\ \mathbf{0} \end{bmatrix}. \tag{6}$$

Once the distance matrices and optimal tour vectors have been padded to the standard size of $N \times N$ and $N \times 1$, respectively, they are used as input instances for training the TSPNet model, described in the TSPNet training process subsection.

C. TSPNet Architecture

The TSPNet model is designed to predict the optimal tour for the TSP instance using the distance matrix as input. The model efficiently predicts near-optimal solutions to TSP instances by leveraging deep learning techniques. The Table I provides a detailed summary of the architecture.

TABLE I. MODEL ARCHITECTURE SUMMARY

Layer (Type)	Output Shape	Param #
Input_layer_1 (InputLayer)	(None, 200, 200)	0
Flatten_1 (Flatten)	(None, 40000)	0
FC1 (Dense)	(None, 64)	2,560,064
Reshape_1 (Reshape)	(None, 1, 64)	0
MHA (MultiHeadAttention)	(None, 1, 64)	132,672
LN1 (LayerNormalization)	(None, 1, 64)	128
FC2 (Dense)	(None, 1, 128)	8,320
Dropout_1 (Dropout)	(None, 1, 128)	0
LN2 (LayerNormalization)	(None, 1, 128)	256
FC3 (Dense)	(None, 1, 40000)	5,160,000
Reshape_2 (Reshape)	(None, 200, 200)	0
Softmax (Softmax)	(None, 200, 200)	0
Total params	**7,861,440**	

The input to the model is a 200×200 distance matrix representing the pairwise distances between cities in a TSP instance with 200 cities. After flattening the matrix, the model applies a fully connected (Dense) layer to embed the data into a lower-dimensional feature space. This is followed by a reshaping step that adjusts the dimensions for further processing with a Multi-Head Attention (MHA) layer, which captures dependencies across different city pairs. Normalization, dense layers, and dropout regularization are employed to stabilize the learning process. Finally, the output is reshaped back into a 200×200 matrix, with a softmax layer of each row of the output matrix to provide probability distributions over the possible arcs between cities to become the predicted adjacency matrix of the TSP problem.

The use of attention mechanisms in deep learning has been explored to enhance the performance of algorithms. Attention mechanisms allow models to focus on relevant parts of the input data, thereby improving the decision-making process in selecting routes [14]. This approach has been successfully applied in various combinatorial optimization tasks, showcasing the versatility and effectiveness of deep learning techniques in tackling NP-hard problems like TSP.

D. TSPNet Training Process

The TSPNet training process begins with compiling the model using the AdamW optimizer [22], which operates with a learning rate of 0.001 and 64 batch sizes, alongside the Categorical Cross-Entropy (CCE) Loss function to address the problem's multi-class classification nature. The model's output is structured as a 2D matrix, where each row corresponds to a city in the traveling salesman problem. The softmax function is applied to convert the logits into probabilities representing the likelihood of each city-to-city connection. The loss function is defined as

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j} \cdot \log(p_{i,j}),$$

where N is the total number of cities (representing the rows of the output matrix), C is the total number of cities in TSP (representing the columns of the output matrix), set at 200, $y_{i,j}$ denotes the true label (either 0 or 1) for the connection between city i and city j (derived from the one-hot encoded label matrix), and $p_{i,j}$ represents the predicted probability for that connection obtained from the softmax output. This summation is performed over all rows and columns of the output matrix.

During training, the model iteratively adjusts its weights to minimize the computed loss, while the accuracy of its predictions is continuously monitored. Upon completion of the training process, the model's output probabilities are transformed back into a tour by selecting the city with the highest probability from each row of the output matrix. The average accuracy of the predicted tour is calculated by comparing it with the actual optimal tour. As illustrated in Fig. 3, the loss incurred during the training of TSPNet, which involved 100,000 training instances and 20,000 instances from a synthetic dataset, stabilizes after 7,146 iterations. The model achieves a maximum accuracy exceeding 98%, as shown in Fig. 4. Following this training phase, the model is evaluated against the TSPLib dataset, with results discussed in the subsequent experimental section.

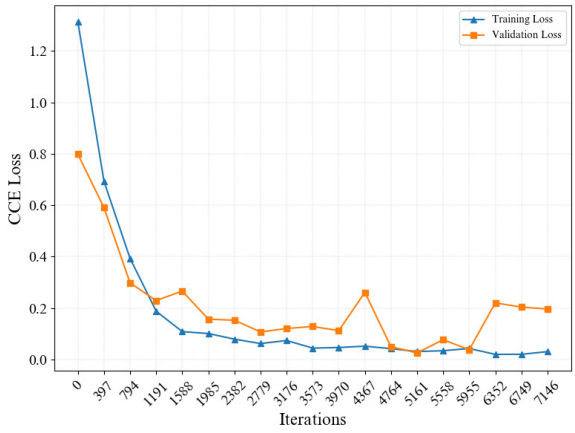


Fig. 3. Categorical cross-entropy loss values during training process.

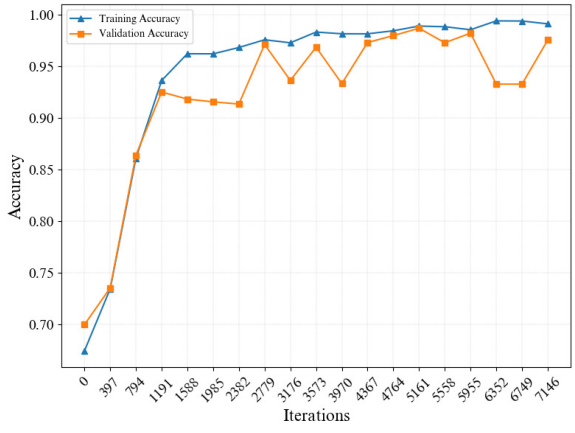


Fig. 4. Accuracy values during training process.

Although the model aims to predict the optimal tour, it

may occasionally yield invalid tours or infeasible solutions. To address this issue, the next subsection will introduce an algorithm designed to generate a valid tour when an optimal solution is not attained, before applying traditional optimization techniques, such as genetic algorithms, to search for the optimal tour.

E. TSPNet Solving Process

This section describes the process of solving the traveling salesman problem using the trained TSPNet model, as illustrated in Fig. 1(b). Given a TSP instance with n nodes, the first step is to generate an $n \times n$ distance matrix by calculating the pairwise distances between the coordinates of each city. This distance matrix is then normalized using min-max normalization before being fed into the pre-trained TSPNet model. The model outputs a zero-padded adjacency matrix, which represents the probabilities of each city being connected to another.

For a clearer understanding, let's consider Fig. 5, which shows a TSP instance with five cities. The predicted output matrix (adjacency matrix) from the TSPNet model is transformed into a predicted tour vector by selecting the city with the highest probability from each row, resulting in the tour vector T, as represented in Eq. (7),

$$\hat{T} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} \text{argmax}\{p_{11}, p_{12}, \ldots, p_{15}\} \\ \text{argmax}\{p_{21}, p_{22}, \ldots, p_{25}\} \\ \text{argmax}\{p_{31}, p_{32}, \ldots, p_{35}\} \\ \text{argmax}\{p_{41}, p_{42}, \ldots, p_{45}\} \\ \text{argmax}\{p_{51}, p_{52}, \ldots, p_{55}\} \end{bmatrix}. \quad (7)$$

This represents an ideal prediction of the optimal tour from the city i^{th} to the j^{th} city. However, when TSPNet encounters more complex instances, the predicted tour vector T may include errors, leading to infeasible or suboptimal solutions. To mitigate this issue, the tour correction algorithm is used to adjust the predicted solution to better approximate the optimal tour while minimizing the total cost or the tour length.

The algorithm consists of four main steps: Step 1 initializes all variables. Step 2 involves separating the predicted tour into subpaths, where each subpath contains unique nodes or cities. In Step 3, missing nodes, those not included in the model's predictions are identified and added as isolated paths containing only one node. After this step, the algorithm will have several subpaths, each containing unique node. Finally, in Step 4, these subpaths are connected by finding the minimum distance between the last node of one subpath and the first node of another, thereby forming a complete and feasible tour. The details of this algorithm are outlined in Algorithm 1.

Once the tour is obtained from Algorithm 1, it can be used as the initial solution for traditional solvers, such as a genetic algorithm. This integration allows the TSPNet prediction to serve as a starting point for other optimization methods, ensuring that the process begins with a feasible tour close to the optimal tour.

The following section presents the experimental results of TSPNet on a benchmark TSP dataset:

Algorithm 1 Find a Single Feasible Tour (With Missing Nodes and Distance Comparison)

Input: `Predicted tour` ($\hat{T}$), Distance matrix (`D`), Number of cities (`n`)
Output: T (feasible tour)

1. Initialize Variables
$T \leftarrow$ An empty feasible tour
$SP \leftarrow$ A set of subpaths
$AN \leftarrow$ A set of total nodes
$N \leftarrow$ A set of existing nodes
$MN \leftarrow$ A set of missing nodes
$MD \leftarrow$ Minimum distance between nodes
$Si \leftarrow$ Start index

 # 2. Split Predicted Tour into Subpaths
for each node i in $\hat{T}$ **do**
    if node i is not in N **then**
        Add node i to N
        Update Si
    else
        if Si is set **then**
            Add the subpath from Si to i to SP
            Reset Si
        end if
    end if
end for

if Si is set **then**
    Add the remaining subpath to SP
end if

 # 3. Add Paths for Missing Nodes into Subpaths
$MN \leftarrow AN - N$
for each node m in MN **do**
    Add node m as a separate subpath to SP
end for

 # 4. Connect Split Subpaths to Form a Tour
Set T to the first subpath and remove it from SP
while Length of $T <$ `n` **do**
    Initialize MD to infinity
    for each subpath in SP **do**
        Calculate the distance from the last node of `T` to the first node of the subpath
        if this distance is smaller than MD **then**
            Update MD and store the index of the subpath
        end if
    end for
    Add the closest subpath to T
    Update the last node in T and remove the selected subpath from SP
end while
return T

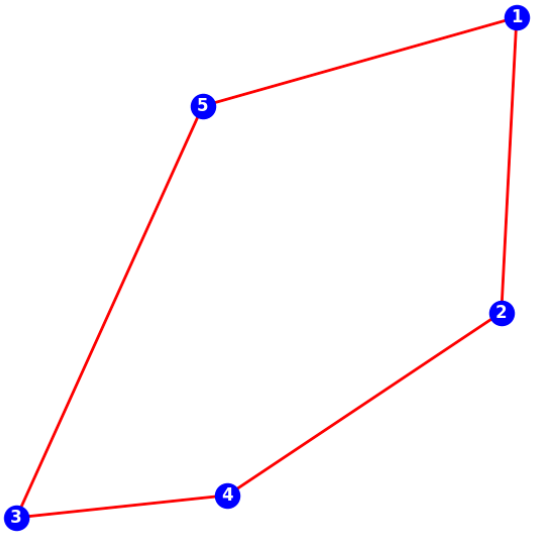


Fig. 5. The optimal tour of a five cities TSP.

VI. Experimental Results

The system architecture is based on an x86_64 platform with an AMD EPYC 7B12 CPU, featuring 8 online CPUs (4 cores per socket with 2 threads each). The deep learning model is trained using TensorFlow [26] on an L4 GPU with 22 GB of VRAM, enhancing processing efficiency. Additionally, the research employs the DEAP [27] library for implementing genetic algorithms and the Gurobi solver for optimization tasks.

A. TSPLIB Dataset

The TSPLIB (Traveling Salesman Problem Library) is a widely used collection of benchmark instances for the traveling salesman problem and its variations, offering a range of instances from small to large sizes. The dataset is formatted in plain text files with problem details such as name, type, dimension, and coordinates or distance matrices. TSPLIB encompasses various problem types, including classic TSP, asymmetric TSP (ATSP), and vehicle routing problems (VRP), allowing researchers to benchmark and compare algorithm performance. The dataset is accessible through its official website and repositories like GitHub.

B. Genetic Algorithm (GA)

The Genetic Algorithm (GA) used in this study was configured with specific parameters to optimize the performance of the TSPLIB dataset. The population size was set to 300 individuals, and the crossover probability was 0.7, allowing a significant proportion of the population to exchange genetic information during each iteration. The mutation probability was set to 0.3, introducing randomness and variation into the population to avoid local optima. A gap threshold of 0.07% was implemented, meaning the algorithm would terminate early if the gap between the best-found solution and the optimal solution was less than 0.07%. The exact optimal distance values were retrieved from the third column of the official TSPLIB web resource and set as the best-known distances for the algorithm to target. The DEAP library was used to run the algorithm, with multiprocessing enabled to parallelize the workload and reduce runtime. Additionally, a local search technique known as 2-opt was applied to refine the best individuals, improving their solutions by eliminating suboptimal edges and enhancing the overall quality of the tour.

The Table II presents a comparison of different TSP solvers, including Mixed-Integer Programming (MIP) solvers (Gurobi and a Genetic Algorithm), and heuristic solvers (TSP-Net and TSPNet combined with GA), evaluated on multiple TSP instances from the TSPLIB dataset. The performance metrics compared include the objective value, the percentage gap from the optimal solution, and the time taken to solve the problems. The gap is calculated as a percentage difference between the best objective value found by Gurobi, GA, TSP-Net, or TSPNet GA and the optimal solution from the official TSPLIB web resource. The formula used for the gap is,

$$\text{Gap} = \frac{100 \times (\text{Best objective value from solver} - \text{Optimal})}{\text{Objective value from solver}}.$$

This formula expresses the relative deviation of the obtained solution from the known optimal solution. A lower gap value indicates a closer match to the optimal solution, with a gap of 0% representing a perfect match.

Gurobi was given an upper bound of 3600 seconds, and it consistently achieves objective values close to the optimal with minimal gaps, though its computation time varies significantly depending on the problem size, ranging from around 0.1 seconds to over 600 seconds, with some instances nearly reaching the time limit. The Genetic Algorithm and TSPNet GA were both given a time limit of 600 seconds. The Genetic Algorithm shows larger gaps and longer computation times compared to Gurobi, with instances such as d198 showing a gap of over 80%. The TSPNet heuristic solver performs exceptionally well, achieving objective values close to the optimal and minimal gaps for most instances, while being significantly faster than the MIP solvers, often solving problems in fractions of a second. The combination of TSPNet with GA slightly improves the objective values and maintains competitive computation times, showing minimal gaps and similar performance to TSPNet in terms of objective values. Overall, TSPNet proves to be a highly effective and fast heuristic solver, delivering accurate results with minimal gaps and outperforming traditional MIP solvers in terms of speed. The combination of TSPNet and GA offers additional benefits by balancing high accuracy with efficient computation times, demonstrating the practicality of heuristic approaches for solving large-scale TSP problems.

VII. Discussion

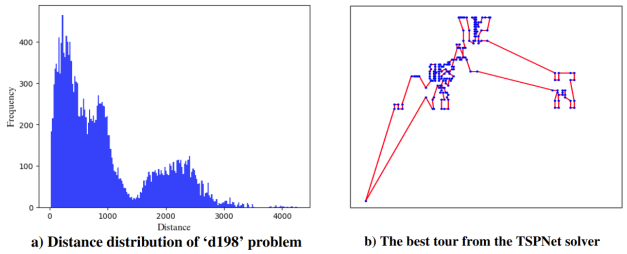
This research presents a deep learning model based on an attention mechanism for solving the Traveling Salesman Problem (TSP), contributing to the growing body of literature exploring innovative approaches to combinatorial optimization. Similar to the study by Kool et al. 2019 [14]. in “Attention, Learn to Solve Routing Problems,” which employs an attention model that utilizes model architecture inspired by the LSTM (Long Short-Term Memory) network to effectively predict optimal tours, this work also leverages attention mechanisms to enhance solution accuracy for TSP. Some experiments’

TABLE II. COMPARISON OF DIFFERENT TSP SOLVERS PERFORMANCE

No.	TSP name	Optimal	MIP solver						Heuristic solver					
			Gurobi			GA			TSPNet			TSPNet GA		
			Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
1	att48	10628	10628	0.00	125	10628	0.00	157	10628	0	0.082	10628	0	0.41
2	ch150	6528	6555	0.41	3600	6804	4.06	600	6530	0.03	0.064	6530	0.03	0.4
3	berlin52	7542	7544	0.03	0.742	7544	0.03	41	7544.36	0.03	0.064	7544	0.03	0.43
4	d198	15780	16440	4.01	3600	91319	82.72	600	16245	2.86	0.073	16245	2.86	600
5	fri26	937	937	0.00	0.579	937	0.00	1.36	937	0	0.061	937	0	0.41
6	ftv33	1286	1286	0.00	0.6	1467	12.34	600	1286	0	0.059	1286	0	0.4
7	ftv35	1473	1473	0.00	0.817	1630	9.63	600	1473	0	0.062	1473	0	0.4
8	ftv47	1776	1776	0.00	1.027	1776	0.00	3	1776	0	0.064	1776	0	0.4
9	ftv38	1530	1530	0.00	0.885	1617	5.38	600	1530	0	0.066	1530	0	0.4
10	ftv64	1839	1839	0.00	3.815	2052	10.38	600	1839	0	0.063	1839	0	0.39
11	ftv55	1608	1608	0.00	1.634	1662	3.25	600	1608	0	0.068	1608	0	0.42
12	ftv44	1613	1613	0.00	0.714	1640	1.65	600	1613	0	0.067	1613	0	0.45
13	bier127	118282	118293	0.01	3600	119340	0.89	600	118293	0.01	0.068	118293	0.01	0.43
14	ch130	6110	6128	0.29	3600	6213	1.66	600	6110	0	0.077	6110	0	0.48
15	burma14	3323	3323	0.00	0.28	3323	0.00	0.89	3323	0	0.087	3323	0	0.4
16	brazil58	25395	25395	0.00	50	25395	0.00	4	25395	0	0.063	25395	0	0.4
17	brg180	1950	1950	0.00	8238	2010	2.99	600	1950	0	0.07	1950	0	0.43
18	bays29	2020	2020	0.00	0.817	2020	0.00	1.28	2020	0	0.063	2020	0	0.43

notable gaps include 1.76% for n = 50 and 4.53% for n = 100, showcasing its competitive performance. This research extends the exploration to more complex instances, specifically focusing on TSP127. The model developed in this study achieved a remarkable objective gap of only 0.01% for the TSP127 instance, demonstrating its capability to deliver highly accurate solutions for large problem sizes. The TSPNet solver has the potential to be more powerful if trained on a large dataset with diverse training distributions.

However, training on a wide range of problems requires significant computational resources. In this study, the synthetic training data were generated from a log-normal distribution as in Fig. 2, which presents some limitations. When data points do not conform to this distribution, the solver may fail to predict the optimal tour, as demonstrated in Fig. 6. The distribution exhibits two peaks in frequency values, which the TSPNet model has not been trained on, representing a limitation of the current model. This issue could be addressed by training the model on a wider variety of distributions in the future.



a) Distance distribution of 'd198' problem b) The best tour from the TSPNet solver

Fig. 6. The best tour predicted by the TSPNet solver for the d198 problem.

VIII. CONCLUSION

Solving a Traveling Salesman Problem (TSP) with iterative processes in a traditional solver is computationally expensive and time-consuming for a large-scale instance. However, this issue can be alleviated by determining the initial optimal or near optimal valid tour to the solver to start. This paper introduces TSPNet, a deep learning-based solver equipped with an attention mechanism for TSP, to effectively predict an optimal or a near-optimal valid tour. TSPNet trained on diverse syntactic data across various distance distributions and it evaluated its performance from the TSPLIB dataset. In cases where the predicted solution is infeasible or does not represent a valid tour, the proposed algorithm adjusts the output to ensure feasibility.

TSPNet consistently generates valid tours that can serve as effective warm starts for traditional solvers, providing a strong initial solution. When evaluated against the TSPLIB benchmark dataset, TSPNet outperformed both Gurobi and a genetic algorithm enhanced with a 2-opt local search, significantly reducing computational time by circumventing the iterative processes typically required in traditional methods. Most results achieved nearly a 0% gap, highlighting the model's effectiveness. However, training on larger problem sizes necessitates substantial computational resources.

Future work will focus on expanding the training dataset to encompass a broader range of distributions and problem instances, thereby enhancing the model's generalization capabilities. Additionally, exploring more efficient training techniques or hybrid approaches that combine deep learning with traditional optimization methods could help reduce resource requirements. Integrating TSPNet with other metaheuristic algorithms represents another promising direction to improve solution accuracy and efficiency for larger-scale TSP instances.

ACKNOWLEDGMENT

This research was supported by the Science Achievement Scholarship of Thailand and the Applied Mathematics and Computational Science Program in the Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, Thailand. The authors acknowledge

using an LLM to enhance the clarity and coherence of this manuscript. However, all scientific content and conclusions are solely the authors' responsibility.

REFERENCES

[1] Cook, William J., et al. The traveling salesman problem: a computational study. Princeton university press, 2011.

[2] I. Bello, "Neural combinatorial optimization with reinforcement learning", 2016. doi: 10.48550/arXiv.1611.09940

[3] W. Liu, R. Wang, T. Zhang, K. Li, W. Li, and H. Ishibuchi,"Hybridization of evolutionary algorithm and deep reinforcement learning for multi-objective orienteering optimization", 2022. doi: 10.48550/arXiv.2206.10464

[4] N. Xu, L. Liu, and Y. Xu, "A novel chaotic neural network with radial basis function and their application to tsp," Appl. Mech. Mater., vol. 151, pp. 532–536, 2012. doi: 10.4028

[5] A. Philip, A. Adio, and O. Kehinde, "A Genetic Algorithm for Solving Travelling Salesman Problem," Int. J. Adv. Comput. Sci. Appl., vol. 2, no. 1, 2011. doi: 10.14569/IJACSA.2011.020104

[6] R. Mahajan and G. Kaur, "Neural networks using genetic algorithms," Int. J. Comput. Appl., vol. 77, no. 14, pp. 6–11, 2013. doi: 10.5120/13549-1153

[7] N. Boyko and A. Pytel, "Aspects of the study of genetic algorithms and mechanisms for their optimization for the traveling salesman problem," International Journal of Computing, pp. 543–550, 2021. doi: 10.47839/ijc.20.4.2442

[8] P. Fu, Y. Wang, and P. Yang, "A particle swarm optimization based on evolutionary game theory for discrete combinatorial optimization," Journal of Convergence Information Technology, vol. 7, no. 21, pp. 369–376, 2012.

[9] S. Sharma, and V. Garg, "Multi colony ant system based solution to traveling salesman problem using opencl," Int. J. Comput. Appl., vol. 118, no. 23, pp. 1–3, 2015. doi: 10.5120/20882-3637

[10] W. Kool, H. Hoof, J. Gromicho, and M. Welling, "Deep policy dynamic programming for vehicle routing problems," Lect. Notes Comput. Sci., vol. 13292, pp. 190–213, 2022. doi: 10.1007/978-3-031-08011-1_14

[11] J. Perera, S. Liu, M. Mernik, M. Črepinšek, and M. Ravber, "A graph pointer network-based multi-objective deep reinforcement learning algorithm for solving the traveling salesman problem," Mathematics, vol. 11, no. 2, p. 437, 2023. doi: 10.3390/math11020437

[12] M. Prates, P. Avelar, H. Lemos, L. Lamb, and M. Vardi, "Learning to solve np-complete problems: A graph neural network for decision tsp," Proc. Conf. AAAI Artif. Intell., vol. 33, no. 01, pp. 4731–4738, 2019. doi: 10.1609/aaai.v33i01.33014731

[13] U. Gunarathna, R. Borovica-Gajić, S. Karunasekara, and E. Tanin, "Solving dynamic graph problems with multi-attention deep reinforcement learning," 2022. doi: 10.48550/arxiv.2201.04895

[14] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!," 2018. doi: 10.48550/arxiv.1803.08475

[15] Y. Jin, Y. Ding, X. Pan, K. He, L. Zhao, T. Qin, et al., "Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem," Proc. Conf. AAAI Artif. Intell., vol. 37, no. 7, pp. 8132–8140, 2023. doi: 10.1609/aaai.v37i7.25982

[16] E. Bellodi, A. Bertagnon, M. Gavanelli, and R. Zese, "Improving the efficiency of euclidean tsp solving in constraint programming by predicting effective nocrossing constraints," Lect. Notes Comput. Sci., vol. 12414, pp. 318–334, 2021. doi: 10.1007/978-3-030-77091-4_20

[17] M. Purnomo, M. Iqbal, and M. Sufa, "Solving multiple routes travelling salesman problem using modified genetic algorithm," Adv. Mat. Res., vol. 576, pp. 718–722, 2012. doi: 10.4028/www.scientific.net/amr.576.718

[18] J. Sarubbi, G. Miranda, H. Luna, and G. Mateus, "A cut-and-branch algorithm for the multicommodity traveling salesman problem," 2008 IEEE International Conference on Service Operations and Logistics, and Informatics, 2008. doi: 10.1109/SOLI.2008.4682823

[19] A. Arigliano, T. Calogiuri, G. Ghiani, and E. Guerriero, "A branch-and-bound algorithm for the time-dependent traveling salesman problem," Networks, vol. 72, no. 3, pp. 382–392, 2018. doi: 10.1002/net.21830

[20] T. Kenea, "Solving shortest route using dynamic programming problem," Indian J. Sci. Technol., vol. 15, no. 31, pp. 1527–1531, 2022. doi: 10.17485/ijst/v15i31.1342

[21] T. Vo, M. Baiou, V. Nguyen, and P. Weng, "Improving subtour elimination constraint generation in branch-and-cut algorithms for the tsp with machine learning," Lect. Notes Comput. Sci., vol. 14286, pp. 537–551, 2023. doi: 10.1007/978-3-031-44505-7_36

[22] I. Loshchilov, and F. Hutter, "Decoupled Weight Decay Regularization, "International Conference on Learning Representations (ICLR)," 2019. 1711.05101.

[23] L. L. C. Gurobi Optimization, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[24] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "Concorde: A code for solving traveling salesman problems," AT&T Labs, 2003. [Online]. Available: http://www.math.uwaterloo.ca/tsp/concorde.html

[25] G. Reinelt, "TSPLIB – A Traveling Salesman Problem Library," Institut für Mathematik, Universität Heidelberg, Heidelberg, Germany, 1991. [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[26] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org

[27] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," J. Mach. Learn. Res., vol. 13, pp. 2171–2175, Jul. 2012. [Online]. Available: https://github.com/DEAP/deap