Learning Insertion Heuristics for the Traveling Salesman Problem via Neural Networks and Black-Box Optimization

Mariusz Kaleta[1] and Tomasz Śliwiński[1]

***Abstract*— This paper introduces a novel neural-based method for solving the Traveling Salesman Problem (TSP) by learning adaptive, single-pass construction heuristic within the Insertion heuristic framework. We design a simple feed-forward neural network that dynamically evaluates candidate cities based on multi-dimensional distance properties, allowing flexible handling of varying problem sizes. The network is trained using a black-box optimization approach with Covariance Matrix Adaptation Evolution Strategy (CMA-ES), bypassing the need for gradient information. To further improve solution quality, we propose a lightweight local refinement technique that perturbs the trained network's weights, effectively exploring nearby heuristic landscapes without restarting the search process. Computational experiments on synthetic datasets with 50, 100, and 200 cities show that our approach consistently outperforms classical Insertion heuristics and generates solutions within 0–2.4% of optimality. The proposed method combines high computational efficiency with adaptability, demonstrating strong potential for extension to more complex TSP variants and practical logistics optimization tasks.**

***Index Terms*— TSP; Insertion heuristics; Construction heuristics; Neural networks; Logistic Optimization;**

I. Introduction

The Traveling Salesman Problem (TSP) is fundamentally linked to logistics applications, serving as a cornerstone for optimizing delivery routes and addressing critical challenges in supply chain management and last-mile delivery. New applications are also emerging, e.g. in humanitarian logistics [1]. In logistics, delivery trucks, couriers, or service vehicles must visit multiple locations (e.g., warehouses, stores, or customer addresses) in the most efficient way possible. The Vehicle Routing Problem, a common logistical challenge, generalizes the TSP to multiple vehicles [2]. Thus, any progress in solving the TSP has significant implications for a wide range of logistics problems.

Let $\mathbf{C} = \{1, 2, \ldots, n\}$ denote a set of n cities. For each pair of cities $(i, j) \in \mathbf{C} \times \mathbf{C}$, the distance between them is given by $d(i, j)$. We consider the Metric TSP, in which $d(i, j)$ satisfies the following properties:

- Non-negativity: $d(i, j) \geq 0$ for all $i \neq j$, and $d(i, j) = 0$ if and only if $i = j$.
- Symmetry: $d(i, j) = d(j, i)$
- Triangle Inequality: $d(i, j) \leq d(i, k) + d(k, j)$

The goal is to find a minimum-length tour visiting each city exactly once, with no subtours (i.e., cycles other than the full tour).

*This work was not supported by any organization

[1]Warsaw University of Technology, Institute of Control and Computation Engineering, Nowowiejska 15/19, 00-665 Warsaw, Poland mariusz.kaleta@pw.edu.pl, tomasz.sliwinski@pw.edu.pl

Being NP-hard, the TSP has been drawing major research attention for decades. Over time, a variety of methods have been developed, which can broadly be classified into exact algorithms and heuristic/metaheuristic approaches.

While exact methods, such as Integer Linear Programming or Dynamic Programming, are limited to small problem sizes [3], [4], advanced techniques combining cutting planes and branch-and-bound — notably the Concorde TSP Solver — can solve standard symmetric TSP instances to optimality with remarkable efficiency [5]. However, these methods have limited potential for extension to more complex TSP variants.

Among heuristic approaches, constructive algorithms such as Nearest Neighbor [6], [7], Greedy [8], and Insertion heuristics [9] are widely used. Improvement heuristics such as 2-Opt and 3-Opt [10] refine initial solutions by local modifications. Metaheuristics are particularly popular due to their superior exploration capabilities, with genetic algorithms being the most applied and ant colony optimization being the most cited [11]. Furthermore, approximation algorithms provide provable performance guarantees; for instance, Christofides' algorithm achieves a solution within 1.5 times the optimal length for metric TSP instances by leveraging minimum spanning trees and matchings [12].

Recent advances in machine learning, particularly neural networks, have opened new avenues for enhancing heuristic design by enabling data-driven, adaptive strategies that learn from problem structure. Both supervised learning [13], [14] and reinforcement learning [15] have been applied to learn construction heuristics. Various neural network architectures have been explored, including Pointer Networks [13], Graph Convolutional Networks [14], and encoder-decoder structures with multi-headed attention [16]. Rather than learning construction heuristics, Da Costa et al. [17] employ deep reinforcement learning to learn a local search (improvement) heuristic based on 2-Opt moves.

Although the TSP is a thoroughly studied problem, there remains a need for methods that can quickly produce high-quality solutions while retaining the flexibility to adapt to problem variants with practical significance in logistics. Our contribution is as follows:

- We introduce a novel approach for learning single-pass construction heuristics based on the Insertion algorithm. We use a simple feed-forward neural network to evaluate candidate cities to be added to the current tour. Unlike most neural-based approaches, our framework is robust to a variable number of inputs.
- Owing to the simplicity of our neural-based evaluator, we are able to train the model using a population-based

approach.

- We extend our single-pass heuristic by exploring the heuristic space through local perturbations of the network weights. In contrast to metaheuristic methods, which restart the search from scratch each time they are run, our approach allows general knowledge to be shared via a pretrained neural network, which serves as the starting point for exploration under the proposed perturbation scheme.

The remainder of this paper is structured as follows. Section II provides details of the proposed neural-based constructive heuristic. Section III discusses a perturbation-based algorithm for further refinement of the solution. Section IV presents numerical experiments, evaluating our approach against standard Insertion Heuristics and optimal solutions. Finally, Section V summarizes our findings.

II. Methods

A. Neural-based constructive heuristic

Insertion heuristics form a class of algorithms that construct a tour starting from a single city with a self-loop. They then iteratively select a city and add it to the tour. We propose a neural-based constructive heuristic that follows this general framework while effectively exploring the space of all possible insertion heuristics.

Let $h \in \mathbf{H}$ represent a specific heuristic from the class of Insertion heuristics. At the beginning of the k-th iteration, the current partial tour is denoted as $s_k \in \mathbf{S}$. The set $\mathbf{S}$ contains all possible partial tours (i.e., all subtours). In iteration k, a new city must be selected and inserted into the current tour. Let $\mathbf{C_k} \subseteq \mathbf{C}$ denote the set of potential decisions (not yet inserted cities) at iteration k.

The Insertion heuristics class encompasses a wide range of methods determining $j \in \mathbf{C_k}$, including the following [9]:

- Nearest Insertion – the next city j to add to the current tour s_k is the one that minimizes $d(i,j)$ over cities i already in the tour s_k.
- Farthest Insertion – similar to Nearest, but city j maximizes $d(i,j)$.
- Cheapest Insertion – selects the city that minimizes the cost $c(i,j,l)$ of inserting city j between cities i and l, where $c(i,j,l) = d(i,j) + d(j,l) - d(i,l)$.

Once city $\hat{j}_k$ is chosen for addition in step k, heuristic h transforms the current partial solution into an updated solution. In general, this transformation is represented by the function:

$$T : (\mathbf{S}, \mathbf{C}) \to \mathbf{S} \tag{1}$$

where $T(s_k, \hat{j}_k)$ adds one city according to decision $\hat{j}_k$ to the current partial tour s_k, producing a new partial or final tour s_{k+1}. Nearest, Farthest, and Cheapest heuristics insert city $\hat{j}_k$ between cities i and l from s_k such that $c(i, \hat{j}_k, l)$ is minimized.

The generic framework of the Insertion heuristic follows these steps:

1. **Initialize**: Set $k = 0$ and initialize the partial solution s_0 as a single city with self-loop.
2. **Iterate**: $k = k + 1$
3. **Select a Decision**: Find the decision $\hat{j}_k$ that maximizes the evaluation function E_h:

$$\hat{j}_k = \arg\max_{j \in C_k} E_h(p_h(s_k, j))$$

where $p_h : (S, C) \to \mathbb{R}^m$ denotes a property function evaluated by

$$E_h(p_h(s_k, j)) : \mathbb{R}^m \to \mathbb{R}^1.$$

4. **Update Solution**: Apply decision $\hat{j}_k$ to extend the current partial solution:

$$s_{k+1} = T(s_k, \hat{j}_k)$$

$$C_{k+1} = C_k \setminus \{\hat{j}_k\}$$

5. **Stopping condition**: If $C_{k+1} \neq \emptyset$, return to Step 2.

We define the property function $p_h(s_k, j)$ in a general form. However, for simple heuristics such as Nearest, Farthest, and Cheapest Insertion, it is scalar — that is, $d(i,j)$ and $c(i,j,l)$ for Nearest/Farthest and Cheapest, respectively. The evaluation function E_h simply minimizes (Nearest/Cheapest) or maximizes (Farthest) the scalar property. Thus, in simple heuristics, E_h is a static rule that does not depend on the current state s_k.

To generalize these approaches, we assume that the property function is an m-dimensional vector. For instance, it may consider distances not only to the nearest or farthest city but to all cities in the current tour. Consequently, the evaluation function E_h can be more complex and depend dynamically on the current partial tour s_k. This general framework includes the classic heuristics such as Nearest, Farthest, and Cheapest Insertion, but also allows for more complex algorithms.

Our approach employs a trained neural network to dynamically select the next city for insertion into the existing tour, predicting the most promising candidate based on learned patterns in the distance matrices. The fundamental idea is to use a neural network as the evaluation function E_h. Let $N_{\bar{w}}$ denote the neural network with optimized weights $\bar{w}$, substituting for E_h in the neural-based variant of the heuristic framework. The neural network learns which city to select next. Following the Insertion framework, it is applied separately to each not-yet-inserted city $j \in C_k$ at each step k, allowing a variable number of candidate cities while maintaining a constant input size for the network. Moreover, since $N_{\bar{w}}$ can process vector inputs, the property function $p_h(s_k, j)$ can be multi-dimensional, enabling multiple factors to be considered when selecting the next city.

Once selected, the city is inserted at the position that minimizes the increase in tour length $c(i, \hat{j}_k, l)$, preserving the cost-efficiency of the Insertion framework while enhancing decision-making through neural guidance.

The procedure for constructing the final solution is depicted in Figure 1, where the role of evaluator E_h is taken over by the trained neural network $N_{\bar{w}}$ with weights $\bar{w}$.

Note that in this approach, once the network is trained, no search of the solution space is performed as in many

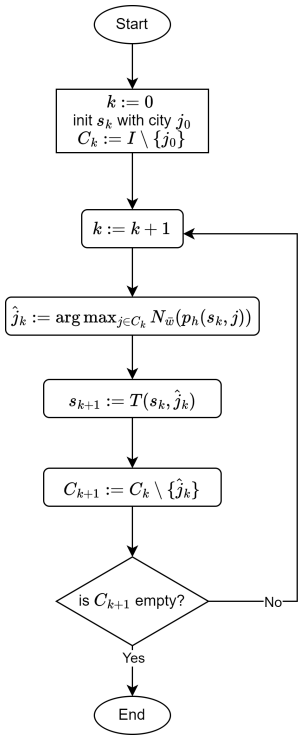

Fig. 1. General scheme of the neural-driven construction heuristic

metaheuristics. Instead, a single pass of the constructive heuristic is used to build the solution.

B. Properties function design

The properties $p_h(s_k, j)$ evaluated by the neural network $N_{\bar{w}}(p_h(s_k, j))$ are intended to reflect the position of the current tour relative to the city j under consideration for insertion. The selection of appropriate properties is a crucial design element that significantly impacts solution quality. The neural network should be provided with a sufficiently rich set of properties to capture different features depending on the current situation and thus achieve good generalization. On the other hand, too many properties make the neural network unnecessarily complex and less efficient.

The choice of properties may depend on the characteristics of the data. Without any prior knowledge about data characteristics, we assume that cities are located uniformly in space. After preliminary experiments, we decided to construct an input vector consisting of the following properties:

Distances to the current tour.
A vector of distances from the examined city $j \in C_k$ to a limited number of cities in the current tour s_k. Altohugh we could technically consider distances to every city in s_k, we instead select a fraction f of cities, evenly spread throughout the tour. This reduces computational effort and keeps input size constant as the tour grows. For simplicity, if the current tour contains fewer cities than needed, some distances are duplicated.

Progress of the algorithm.
The fractional value $p = k/n$ indicates how advanced is the construction of the final tour.

C. Network architecture

We use a simple architecture for the network $N_{\bar{w}}$, depicted in Figure 2. The goal is to achieve high network efficiency, enabling a population-based training procedure in which the network acts as an evaluator within an individual. More expressive architectures, such as transformer-based, require significantly more computation resources and would slow the search process. The network has a four-layer feed-forward architecture with a relatively small number of inputs and a single output.

Let $m = \lceil f \cdot n \rceil$ be the number of cities whose distances are included in the input. After preliminary experiments, we set f to 20% for problems with $n \in \{50, 100\}$ and to 15% for problems with $n = 200$. Thus, the number of network inputs is either $\lceil 0.2 \cdot n \rceil + 1$ or $\lceil 0.15 \cdot n \rceil + 1$, depending on the problem size.

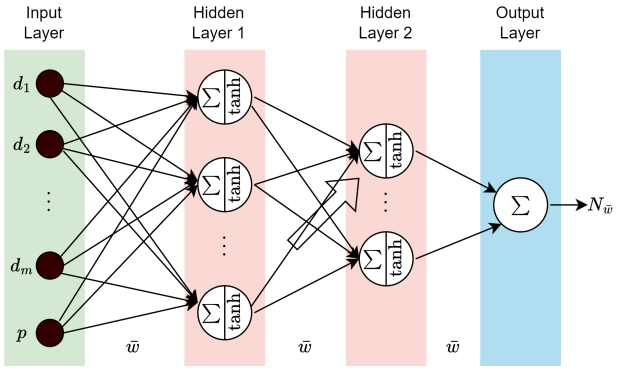

Fig. 2. Architecture of the network $N_{\bar{w}}$.

The cities used to compute distances are evenly distributed along the current tour s_h. Figure 3 presents an example for $n = 40$ cities, where red crosses denote cities already in the current tour, black dots represent candidate cities, and the green dot is the city being evaluated for insertion. Assuming $f = 20\%$, distances to $m = 0.2 \times 40 = 8$ evenly spaced cities are indicated by black dashed lines and form part of the network input (together with p).

In more sophisticated scenarios, for instance when regions of higher city density are identified, additional information about the surroundings of the evaluated city could be incorporated — for example, distances to the city's closest neighbors.

Optionally, we consider sorting the distances in non-decreasing order, so that the neural network is consistently exposed to the closest and farthest distances on the same inputs. In our experiments, we report results for both sorted and unsorted cases.

As a result of experimentation, we set the sizes of the first and second hidden layers as follows:

- 16 and 8 neurons, for problems with 50 cities,
- 24 and 12 neurons, for problems with 100 and 200 cities.

Both the hidden layers and the output layer are equipped with bias terms. The total number of network parameters (weights) is 337, 841, and 1,080 for problems with 50, 100, and 200 cities, respectively.

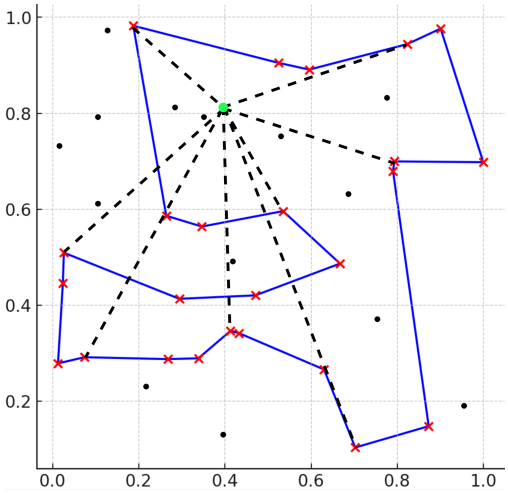

Fig. 3. Illustration of distance evaluation for a candidate city.

For better performance, we use an approximate *tanh* function. In the range $[-2.779, 2.779]$, the function is approximated by a fourth-degree polynomial; outside this range, it is approximated by a linear function with slope 0.01 (see Figure 4). The piecewise approximation of $tanh(x)$ is defined as follows:

$$tanh(x) \approx \begin{cases} (x+2.779)\cdot 0.01 - 0.998, \text{ for } x < -2.779 \\ -1 + \left(\frac{x+3.5}{3.5}\right)^4, \text{ for } -2.779 \leq x < 0 \\ 1 - \left(\frac{x-3.5}{3.5}\right)^4, \text{ for } 0 \leq x < 2.779 \\ (x-2.779)\cdot 0.01 + 0.998, \text{ for } 2.779 \leq x \end{cases}$$

D. Network training with black-box optimization

Since the neural network is embedded within a combinatorial construction heuristic and there is no explicit way to determine correct outputs, typical backpropagation learning schemes cannot be applied. However, black-box optimization methods for neural network training have gained significant attention in the machine learning community [18], [19], [20]. Among various evolutionary strategies, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is widely regarded as a state-of-the-art derivative-free optimization method. It is particularly suitable for black-box optimization, as it relies solely on function evaluations [21].

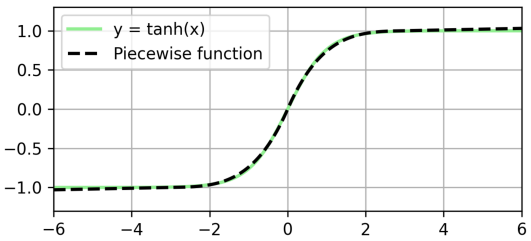

Fig. 4. Approximation of the $tanh(x)$

The algorithm iteratively processes a population of individuals, where each individual represents a set of neural network weights and, consequently, a distinct neural network. Each individual from the population is evaluated using the black-box function, meaning that the associated neural network is embedded as $N_{\bar{w}}$ in our constructive heuristic (Figure 1). This evaluation is performed on a batch of training instances created by randomly selecting 50 problem instances from a total of 1,000,000 available instances. As a result, we obtain an average tour length, which estimates how well an individual performs. CMA-ES uses these evaluations, along with estimations of the mean and covariance matrix, to generate an offspring population and update the distribution parameters. The sole objective of the optimization process is to minimize the tour length.

Every 10th iteration, the best solution found in that iteration is evaluated against a predefined validation set of 10,000 problems. The best-performing solution on the validation set so far is stored as the final solution of the algorithm.

III. Local Refinement of the Learned Heuristic

Our neural-driven constructive heuristic executes a single pass of the algorithm, progressively incorporating new cities into a growing partial tour. This design prioritizes computational speed, making it highly efficient, but leaves potential for refinement when addressing NP-hard challenges. Typically, metaheuristics such as simulated annealing, genetic algorithms, tabu search, ant colony optimization, and particle swarm optimization, paired with local search methods, are used to refine initial solutions by exploring the solution space. Here, we propose a novel metaheuristic paradigm that explores possibilities in a unique manner: rather than directly probing the solution space, this method investigates the landscape of heuristics. Moreover, it capitalizes on insights gained from previously solved problems, assuming that those problems share some commonality.

It can be observed that while $N_{\bar{w}}$ evaluates several choices, it may assign nearly equivalent scores. By considering a range of high-scoring alternatives at each step, we could generate multiple tour paths, some of which might outperform the single best choice. This exploration could take various forms. For example, a random selection process could pick among top candidates, with probabilities tied to their relative scores. Alternatively, a tailored Beam Search could track a fixed number of promising partial tours [22], [23], [24]. However, we propose a different approach that hinges on the insight that slight tweaks to the neural network's weights can alter the ranking of top decisions at each step. The closer the scores of leading options, the easier it is to shift to an alternative by adjusting weights. Perturbing these weights thus enables us to probe a range of promising heuristics defined by specific weight configurations.

Initially, we tune the neural network's weights using CMA-ES to enhance the constructive heuristic's performance

across a broad set of training instances. However, to further refine the heuristic for a specific instance, we apply CMA-ES again, this time focusing solely on that instance. Starting from the weights trained on the broader set, the algorithm conducts a guided evolutionary search through the weight space, seeking the optimal heuristic for that particular problem. This approach is notably efficient: it not only solves the specific instance within the black-box framework but also builds on prior knowledge of promising optimization paths, amplifying the process's effectiveness. Classic metaheuristics such as Simulated Annealing, VNS, etc., perform the search from scratch, neglecting previous searches. Contrary to this, our neural-driven algorithm exhibits aversion to search spaces that historically did not result in good solutions.

IV. Computational experiments

The proposed neural-driven heuristic was evaluated on synthetic instances with 50, 100, and 200 cities. Each instance was generated by randomly placing cities according to a uniform distribution over a 2D square with side length 1; that is, each X and Y coordinate of a city was sampled uniformly from $[0,1]$. Cartesian distances were then computed between all pairs of points.

We tested two versions of the heuristic: with and without distance sorting. As our neural-driven heuristic extends the Insertion Heuristic, we compared it to the classic Insertion Heuristic in two versions, where either the nearest or the farthest city is selected for inclusion in the tour (we omit results for the Cheapest Insertion variant, as it regularly performed worse or very close to either the Nearest or Farthest Insertion heuristic.)

The algorithm was implemented in C++. The neural network implementation leveraged Eigen, a C++ template library for linear algebra (http://eigen.tuxfamily.org). For black-box optimization, we used the CMA-ES library developed at the Laboratory for Computer Science, Université Paris-Sud (https://github.com/CMA-ES/libcmaes), authored by Emmanuel Benazera and supported by the coauthor of the original method, Nikolaus Hanse. The population size was set to 192 individuals (aligned with the CPU's 12 cores and 24 threads)), and the initial step-size parameter was set to $\sigma = 0.4$. As the stopping condition, we limited the number of black-box function evaluations to 300,000, which is equivalent to 1,560 iterations. We used the Sep-CMA-ES variant of CMA-ES, which employs a diagonal covariance matrix to achieve linear computational complexity [25]. All experiments were conducted on a PC-class computer equipped with a 12-core processor operating at an average clock speed of 4.3 GHz.

The average tour lengths over 100 random instances are presented in Table I for the following algorithms: Insertion Heuristic (IH), both Nearest and Farthest versions; our single-pass neural-based Constructive Heuristic (CH); and neural-based Constructive Heuristic with refinement ($\text{CH}_{\text{refined}}$), with both sorted and non-sorted distance variants. The refinement algorithm was $\text{CH}_{\text{refined}}$ run for 30s. We also include optimal tour lengths computed using the Concorde solver for reference.

TABLE I

Tour length for different algorithms.

n	IH		CH		$\text{CH}_{\text{refined}}$		Opt.
	near.	farth.	sort	no sort	sort	no sort	
50	6.497	6.055	5.898	5.930	5.659	5.659	5.659
100	9.073	8.688	8.258	8.346	7.758	7.763	7.737
200	12.727	12.533	11.678	11.739	10.941	11.000	10.683

The convergence behavior of the proposed constructive heuristic with refinement is shown in Tables II and III, for the versions with and without distance sorting, respectively. The tables report the shortest tour lengths obtained within specified time limits: 1, 2, 5, 10, and 30 seconds. For comparison, the average computation time of the Concorde algorithm is 61 seconds for $n = 200$.

TABLE II

Tour length vs. computation time (with distance sorting).

n	1s.	2s.	5s.	10s.	30s.	Opt.
50	5.6591	5.6591	5.6591	5.6591	5.6591	5.6589
100	7.8026	7.7906	7.7730	7.7615	7.7584	7.7372
200	11.0914	11.0704	11.0384	11.0112	10.9411	10.6831

TABLE III

Tour length vs. computation time (without distance sorting).

n	1s.	2s.	5s.	10s.	30s.	Opt
50	5.6594	5.6594	5.6594	5.6594	5.6594	5.6589
100	7.8089	7.7910	7.7724	7.7644	7.7631	7.7372
200	11.1678	11.1407	11.1034	11.0694	11.0000	10.6831

Tables IV and V report the number of tours constructed during the heuristic search under the corresponding time limits, again for the sorted and non-sorted versions.

TABLE IV

Number of checked solutions (distance sorting).

n	1s.	2s.	5s.	10s.	30s.
50	32,059	62,533	16,3016	309,868	885,993
100	6,104	12,431	30,114	59,050	172,525
200	1,099	2,003	4,753	9,752	28,449

We also conducted tests where a model trained on smaller problem sizes was applied to larger instances. Specifically, models trained on problems with 50, 100, and 200 cities were applied to problems with 100, 200, and 400 cities, respectively. The results are presented in Table VI.

V. Conclusions

We presented a novel neural-driven heuristic for the Traveling Salesman Problem based on learning within the framework of Insertion Heuristics. Computational experiments demonstrate that each variant of our method consistently outperforms standard Insertion Heuristics. The refined version of our constructive heuristic ($\text{CH}_{\text{refined}}$) with distance

TABLE V

NUMBER OF CHECKED SOLUTIONS (WITHOUT DISTANCE SORTING).

n	1s.	2s.	5s.	10s.	30s.
50	39,319	68,495	177,833	341,800	1,045,890
100	7,805	15,503	38,583	77,532	229,114
200	1,748	3,313	8,233	16,376	47,733

TABLE VI

TOUR LENGTH, OBTAINED WITH A MODEL TRAINED ON REDUCED PROBLEM SIZE (WITH AND WITHOUT DISTANCE SORTING).

n	CH		$CH_{refined}$		Opt.
	sort	no-sort	sort	no-sort	
100	8.348	8.396	7.784	7.794	7.737
200	11.679	11.714	10.970	11.016	10.683
400	16.531	16.662	15.802	15.984	14.867

sorting achieves improvements ranging from 6.5% for 50-city instances to 12.7% for 200-city instances. Furthermore, our solutions remain within 0–2.4% of the optimal tour lengths.

Sorting the input distances yields minor improvements (0.5%–1.1%), while the perturbation-based refinement step offers more substantial gains (4.1%–6.3%). These results validate the effectiveness of combining simple feed-forward neural networks with black-box optimization and local refinement strategies. Additionally, we showed that models trained on smaller instances can be effectively applied to larger problem sizes, demonstrating the algorithm's generalization capability. To validate our algorithm further, we aim to provide an ablation study as the next research step.

We believe this approach holds strong potential for extension to more complex TSP variants and related combinatorial optimization problems, such as the Rich Vehicle Routing Problem (RVRP), particularly in logistics and operational research applications.

REFERENCES

[1] P. Praneetpholkrang and S. Kanjanawattana, "Application of traveling salesman problem for logistics relief effort," in *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2023, pp. 483–487.

[2] F. A. Houssein, V. F. Kostyukov, and I. D. Evdokimov, "A method for solving the multi-traveling salesman problem based on reducing the size of the solution space," in *2024 10th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2024, pp. 1729–1733.

[3] M. Sniedovich, "A dynamic programming algorithm for the traveling salesman problem," *SIGAPL APL Quote Quad*, vol. 23, no. 4, p. 1–3, June 1993. [Online]. Available: https://doi.org/10.1145/173834.173835

[4] E. Vercesi, S. Gualandi, M. Mastrolilli, and L. M. Gambardella, "On the generation of metric tsp instances with a large integrality gap by branch-and-cut," *Mathematical Programming Computation*, vol. 15, no. 2, pp. 389–416, Jun 2023. [Online]. Available: https://doi.org/10.1007/s12532-023-00235-7

[5] D. L. Applegate, R. E. Bixby, V. Chvatál, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006. [Online]. Available: http://www.jstor.org/stable/j.ctt7s8xg

[6] D. Rosenkrantz, R. Stearns, and P. II, "An analysis of several heuristics for the traveling salesman problem," *SIAM J. Comput.*, vol. 6, pp. 563–581, 09 1977.

[7] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: A case study in local optimization," in *Local Search in Combinatorial Optimization*, E. Aarts and J. Lenstra, Eds. New York: John Wiley & Sons, 1997, pp. 215–310.

[8] J. J. Bentley, "Fast algorithms for geometric traveling salesman problems," *ORSA Journal on Computing*, vol. 4, no. 4, pp. 387–411, 1992.

[9] F. R. J. S. Dantzig, G., "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[10] S. Lin, "Computer solutions of the traveling salesman problem," *The Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.

[11] B. Toaza and D. Esztergár-Kiss, "A review of metaheuristic algorithms for solving tsp-based scheduling optimization problems image 1," *Applied Soft Computing*, vol. 148, p. 110908, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494623009262

[12] C. Christofides, "Worst-case analysis of a new heuristic for the traveling salesman problem," *Technical Report, Graduate School of Industrial Administration, Carnegie Mellon University*, vol. 1, pp. 1–38, 1976.

[13] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf

[14] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *CoRR*, vol. abs/1906.01227, 2019. [Online]. Available: http://arxiv.org/abs/1906.01227

[15] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ByxBFsRqYm

[16] H. Cheng, H. Zheng, Y. Cong, W. Jiang, and S. Pu, "Select and optimize: Learning to solve large-scale tsp instances," in *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, F. Ruiz, J. Dy, and J.-W. van de Meent, Eds., vol. 206. PMLR, 25–27 Apr 2023, pp. 1219–1231. [Online]. Available: https://proceedings.mlr.press/v206/cheng23a.html

[17] P. da Costa, J. Rhuggenaath, Y. Zhang, A. Akcay, and U. Kaymak, "Learning 2-opt heuristics for routing problems via deep reinforcement learning," *SN Computer Science*, vol. 2, no. 5, p. 388, Jul 2021. [Online]. Available: https://doi.org/10.1007/s42979-021-00779-2

[18] J. Arabas and D. Jagodziński, "Toward a matrix-free covariance matrix adaptation evolution strategy," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 84–98, 2020.

[19] P. Carvalho, N. Lourenço, and P. Machado, "How to improve neural network training using evolutionary algorithms," *SN Computer Science*, vol. 5, no. 6, p. 664, Jun 2024. [Online]. Available: https://doi.org/10.1007/s42979-024-02972-5

[20] D. Jagodziński, Ł. Neumann, and P. Zawistowski, "Deep neuroevolution: Training neural networks using a matrix-free evolution strategy," in *Neural Information Processing*, T. Mantoro, M. Lee, M. A. Ayu, K. W. Wong, and A. N. Hidayanto, Eds. Cham: Springer International Publishing, 2021, pp. 524–536.

[21] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[22] I. Sabuncuoglu and M. Bayiz, "Job shop scheduling with beam search," *European Journal of Operational Research*, vol. 118, no. 2, pp. 390–412, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221798003191

[23] S. Wang, Z. Bing-Hai, and X. Li-Feng, "A filtered-beam-search-based heuristic algorithm for flexible job-shop scheduling problem," *International Journal of Production Research - INT J PROD RES*, vol. 46, pp. 3027–3058, 06 2008.

[24] E. Birgin, J. Ferreira, and D. Ronconi, "List scheduling and beam search methods for the flexible job shop scheduling problem with sequencing flexibility," *European Journal of Operational Research*, vol. 247, no. 2, pp. 421–440, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221715005378

[25] R. Ros and N. Hansen, "A simple modification in cma-es achieving linear time and space complexity," in *Parallel Problem Solving from Nature – PPSN X*, G. Rudolph, T. Jansen, N. Beume, S. Lucas, and C. Poloni, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 296–305.