



Faculty of Electrical Engineering  
Department of Cybernetics


Master's thesis

# Learning Based Solution to Routing Problems

Bc. Petra Fridrichová

May, 2023

Supervisor: prof. Ing. Jan Faigl, Ph.D.



*“For the protection of wisdom is like the protection of money, and the advantage of knowledge is that wisdom preserves the life of him who has it.”*

– Ecclesiastes 7:12

*„Být zaštitěn moudrostí je jako být zaštitěn penězi, poznání moudrosti však má výhodu: ty, kdo ji mají, drží naživu.“*

– Kazatel 7:12

## I. Personal and study details

Student's name: **Fridrichová Petra** Personal ID number: **474384**  
Faculty / Institute: **Faculty of Electrical Engineering**  
Department / Institute: **Department of Cybernetics**  
Study program: **Open Informatics**  
Specialisation: **Computer Vision and Image Processing**

## II. Master's thesis details

Master's thesis title in English:

**Learning Based Solution to Routing Problems**

Master's thesis title in Czech:

**ešení sm rovacích problém založené na u ení**

Guidelines:

1. Familiarize with neural network (NN) and deep learning (DL) approaches to routing problems [1–4], and survey existing approaches.
2. Prospect existing methods for generalization to non Euclidean (graph based) routing problems, time windows, and multi vehicle instances.
3. Propose a solution to the selected routing problem based on reusing existing solutions using a suitable representation of the instances, such as a graph neural network encoder of the instance.
4. Implement and evaluate the proposed approach combined with existing population based or combinatorial metaheuristic methods.

Bibliography / sources:

- [1] Joshi, C.K., Cappart, Q., Rousseau, LM. et al.: Learning the travelling salesperson problem requires rethinking generalization. *Constraints* 27, 70–98 (2022). <https://doi.org/10.1007/s10601-022-09327-y>  
[2] Kool, W., van Hoof, H., Welling, M.: Attention, Learn to Solve Routing Problems! *ICLR* 2019.  
[3] Y. Wu, W. Song, Z. Cao, J. Zhang and A. Lim: Learning Improvement Heuristics for Solving Routing Problems, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 9, pp. 5057-5069, Sept. 2022. <https://doi.org/10.1109/TNNLS.2021.3068828>  
[4] B. Li, G. Wu, Y. He, M. Fan and W. Pedrycz: An Overview and Experimental Study of Learning Based Optimization Algorithms for the Vehicle Routing Problem, *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 7, pp. 1115-1138, 2022. <https://doi.org/10.1109/JAS.2022.105677>

Name and workplace of master's thesis supervisor:

**prof. Ing. Jan Faigl, Ph.D. Artificial Intelligence Center FEE**

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **14.09.2022** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **19.02.2024**

\_\_\_\_\_  
prof. Ing. Jan Faigl, Ph.D.  
Supervisor's signature

\_\_\_\_\_  
prof. Ing. Tomáš Svoboda, Ph.D.  
Head of department's signature

\_\_\_\_\_  
prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

### III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

\_\_\_\_\_  
Date of assignment receipt

\_\_\_\_\_  
Student's signature



## Declaration

I declare that the presented work was developed independently and that I have listed all sources of the information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 26, 2023

Petra Fridrichová



## Acknowledgement

I am deeply grateful to Professor Jan Faigl who has always provided insightful suggestions and guidance to improve the quality and complexity of this work. His unwavering support and willingness to address any issues have been invaluable. I would also like to thank Jindřiška Deckerová and Petr Čížek for their patience and responsiveness in answering my countless questions. Their help was crucial in overcoming the challenges I encountered. Finally, I would like to express my sincere gratitude to my husband and family for their unwavering belief in my abilities and their constant encouragement, pushing me to continue writing even when I felt overwhelmed.

## Abstract

The Traveling Salesman Problem (TSP) is a de facto standard benchmark in combinatorial optimization with many existing optimal and heuristic solvers. Nowadays, machine learning approaches have been proposed to address the TSP; however, their performance cannot compete with state-of-the-art methods. In the thesis, a novel hybrid method combining machine learning based on the attention model and the existing unsupervised learning-based GSOA method is proposed leveraging the advantages of both approaches. The hybrid approach initializes a construction heuristic based on retrieving the most similar instance from already pre-solved solutions. The retrieved similar solution shows that GSOA can find better solutions than solely using the attention model and GSOA method.

**Keywords:** traveling salesman problem, neural networks, attention model, GSOA.

## Abstrakt


Na problém obchodního cestujícího spadající do kombinatorické optimalizace existuje mnoho optimálních a heuristických přístupů. V současné době byly pro jeho řešení navrženy různé metody strojového učení, které však svými výsledky nemohou konkurovat nejmodernějším algoritmům. V této práci je navržena nová hybridní metoda kombinující strojové učení založené na modelu pozornosti a metodě GSOA, která využívá výhody obou přístupů. Na základě nejpodobnější instance z již předem vyřešených řešení hybridní přístup inicializuje konstruktivní heuristiku. Využití podobných řešení ukazuje, že metoda GSOA dokáže najít lepší řešení než pouhé použití modelu pozornosti nebo metody GSOA.

**Klíčová slova:** problém obchodního cestujícího, neuronové sítě, model pozornosti, GSOA.

# Used Abbreviations

<b>AM</b>	Attention Model
<b>CNN</b>	Convolutional Neural Network
<b>DL</b>	Deep Learning
<b>FCNN</b>	Fully Convolutional Neural Network
<b>FSC</b>	Feasible Solution Creator
<b>GA</b>	Genetic Algorithm
<b>GNN</b>	Graph Neural Network
<b>GRASP</b>	Greedy Randomized Adaptive Search Procedures
<b>GSOA</b>	Growing Self-Organizing Array
<b>HNN</b>	Hopfield Neural Network
<b>LK</b>	Lin–Kernighan heuristic
<b>LKH</b>	Lin-Kernighan-Helsgaun heuristic
<b>LNS</b>	Large Neighborhood Search
<b>ML</b>	Machine Learning
<b>NN</b>	Neural Network
<b>NLNS</b>	Neural Large Neighborhood Search
<b>RVNS</b>	Randomized Variable Neighborhood Search
<b>SA</b>	Simulated Annealing
<b>SGN</b>	Sparse Graph Network





<b>SOM</b>	Self-Organizing Map
<b>SoS</b>	Set of Solutions
<b>TSP</b>	Traveling Salesmen Problem
<b>UHGS</b>	Unified Hybrid Genetic Search
<b>VNS</b>	Variable Neighborhood Search
<b>VRP</b>	Vehicle Routing Problem

# Used Symbols

$n$	Number of locations
$\mathbf{v}_i$	$i$ -th location
$\ \mathbf{v}_a, \mathbf{v}_b\ $	Euclidean distance between locations $\mathbf{v}_a$ and $\mathbf{v}_b$
$\Sigma$	Permutation of location indexes
$\sigma_1, \dots, \sigma_n$	Location indexes of the path, $\sigma_i \in \{1, \dots, n\}$
$\mathcal{L}(\Sigma)$	Length of TSP path determined by $\Sigma$
$\mathcal{L}^*$	Optimal length of the TSP instance

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Related Work</b>	<b>5</b>
2.1	Convolutional Neural Network . . . . .	6
2.2	Attention Model . . . . .	7
2.3	Graph Neural Network . . . . .	7
2.4	NeuroLKH . . . . .	9
2.5	Neural Large Neighbourhood Search . . . . .	10
2.6	Conclusion on Solving the TSP by Neural Networks . . . . .	10
<b>3</b>	<b>Problem Statement</b>	<b>11</b>
3.1	Euclidean Traveling Salesman Problem . . . . .	11
3.2	Hybrid Approach . . . . .	12
3.2.1	Research Question . . . . .	12
<b>4</b>	<b>Background</b>	<b>13</b>
4.1	Encoder-Decoder Architecture . . . . .	13
4.2	Growing Self-Organizing Array . . . . .	15
4.3	Variable Neighborhood Search . . . . .	16
<b>5</b>	<b>Proposed Hybrid Approach</b>	<b>19</b>
5.1	Set of Solutions Concept . . . . .	19
5.2	Feature Extractor . . . . .	20
5.3	Feasible Solution Creator . . . . .	21
5.3.1	Greedy Construction of Feasible Solution . . . . .	21
5.3.2	GSOA-based Construction of Feasible Solution . . . . .	22
5.4	Improvement Heuristic . . . . .	22
<b>6</b>	<b>Results</b>	<b>25</b>
6.1	Performance of the Existing NN-based Solvers . . . . .	26
6.2	Influence of using SoS Instances . . . . .	28

6.3	Influence of Feasible Solution Creators . . . . .	28
6.4	Influence of the Initialization to the VNS-based Improvement Heuristic	29
6.5	Discussion . . . . .	30
<b>7</b>	<b>Conclusion</b>	<b>33</b>
	<b>Bibliography</b>	<b>35</b>
<b>A</b>	<b>Detailed Results</b>	<b>39</b>

# List of Figures

2.1	Convolutional neural network for the TSP . . . . .	6
2.2	Attention model decoder . . . . .	7
2.3	Graph neural network for the TSP . . . . .	8
2.4	Schema of the NeuroLKH . . . . .	9
3.1	Proposed idea of the hybrid pipeline . . . . .	12
4.1	Encoder-decoder architecture . . . . .	14
4.2	GSOA ring adaptation . . . . .	16
4.3	VNS operators . . . . .	17
5.1	Concept of input instance and set of solutions . . . . .	20
5.2	Graph embedding as feature vector . . . . .	20
5.3	Greedy Feasible Solution Creator . . . . .	21
6.1	Example instances . . . . .	26
6.2	Aggregated results for existing NN-based approaches . . . . .	27
6.3	Retrieved similar instances . . . . .	29
6.4	Aggregated results for the examined feasible solution creators . . . . .	30
6.5	Influence of initializations to the VNS-based solution improvement . . . . .	31

# List of Tables

6.1	Aggregated results for SoS <sub>Random</sub> and SoS <sub>TSPLIB</sub> . . . . .	28
A.1	Performance of the existing NN-based approaches on the Random instances – Part 1 . . . . .	40
A.2	Performance of the existing NN-based approaches on the Random instances – Part 2 . . . . .	41
A.3	Performance of the existing NN-based approaches on the TSPLIB instances – Part 1 . . . . .	42
A.4	Performance of the existing NN-based approaches on the TSPLIB instances – Part 2 . . . . .	43
A.5	Performance of the proposed hybrid approaches on the Random instances – Part 1 . . . . .	44
A.6	Performance of the proposed hybrid approaches on the Random instances – Part 2 . . . . .	45
A.7	Performance of the proposed hybrid approaches on the TSPLIB instances – Part 1 . . . . .	46
A.8	Performance of the proposed hybrid approaches on the TSPLIB instances – Part 2 . . . . .	47



# List of Algorithms

1	Variable Neighborhood Search (VNS) . . . . .	16
2	<code>localSearch(V, <math>\Sigma</math>)</code> . . . . .	23

# Introduction

Routing problems are combinatorial optimization tasks to determine a cost-efficient (shortest) path to visit a given set of locations. Probably the most famous routing problem is the well-known Traveling Salesmen Problem (TSP). In the TSP, a traveling salesman has to visit exactly once each location of a given set of locations and return to the starting one such that the total length of the tour visiting the locations is the shortest possible. The TSP is known to be NP-hard [1], and many methods and approaches have been proposed [2]. The existing solutions to the TSP can be classified into three basic classes of methods: optimal solvers, approximation algorithms, and heuristic solutions.

Optimal solvers are based on general solution techniques such as branch-and-bound, branch-and-cut, or integer linear programming. Here, it is worth noting that a highly tuned implementation of the optimal solver to the TSP called **Concorde** is available for academic research use [3]. It is based on the work of Applegate, Bixby, Chvátal, and Cook [1] that is capable of solving instances up to 85 900 locations. Concorde is based on several mathematical techniques, such as cutting planes and branch-and-cut. However, due to the NP-hardness of the TSP, finding an optimal solution is computationally demanding, and it becomes quickly intractable with the increasing size of the instance.

Approximation algorithms provide solutions with a provable guarantee of the solution quality and run in polynomial computational time. The fundamental approximation algorithm to the TSP is Christofides' algorithm [4] based on finding a minimum spanning tree and perfect matching. It is also called a  $3/2$ -approximation algorithm because all found solutions are shorter than  $3/2$  of the optimal length for a given TSP instance. Note that using the minimum spanning tree yields solutions no worse than two times longer than the optimal solution.

Since optimal solvers are demanding and approximation algorithms provide solutions with relatively high approximation factors for many practical cases, heuristic approaches are studied as they show the best trade-off between the solution quality and computational requirements. Even though the solvers do not have any guarantees on the solution, it usually provides relatively high-quality solutions quickly. Nowadays, from a practical point of view, there is no better way to handle similar routing problems on a market level, specifically with anytime solvers that provide the first solution very quickly and improve it with more computational



time available. With the growing number of locations, it is not possible to rely on optimal solvers that are too demanding to provide even an initial solution. Therefore, combinatorial metaheuristics such as the Genetic Algorithm (GA) [5], Simulated Annealing (SA) [6], Variable Neighborhood Search (VNS) [7], and Greedy Randomized Adaptive Search Procedures (GRASP) [8] have been employed in solving the TSP and became the choice for many businesses. In cases where an optimal solution is not required, metaheuristics represent a general framework to find solutions with a suitable trade-off between the solution quality and computational costs.

In addition to general combinatorial metaheuristics, we can further distinguish two classes of standard heuristics. The first methods are construction heuristics to determine a feasible solution. A feasible solution is a tour that visits all the given locations in not necessarily the shortest possible way. The class of construction heuristics includes a random permutation, greedy (cheapest) insertion, and nearest neighborhood, to name a few [2]. The second class of heuristics represents improving procedures based on local or global search techniques. One of the earliest local search-based, yet powerful, improving heuristics is Lin–Kernighan heuristic (LK) introduced in 1973 [9]. It obtains an initial feasible solution first, then employs an improving method to improve the path and possibly escapes local optima. However, its efficient implementation has been introduced several decades later by K. Helsgaun in 2000 as the Lin–Kernighan–Helsgaun heuristic (LKH) [10], which further introduced additional extensions. The LKH includes several improvements, optimizations, and various heuristics to guide the search process over the LK. Similarly to Concorde, the LKH is available for academic and non-commercial use [11].

Despite the well-established solvers and approaches to the TSP, the current approaches aim to exploit the capabilities of Machine Learning (ML), Deep Learning (DL), and Neural Network (NN) in a solution of the combinatorial optimization problems. Due to the ability of the learning-based methods to scale, generalize, adapt to a learned domain, and provide abstractions, these techniques are also investigated within the context of the routing problems, and specifically, the TSP, which can be considered as the benchmark for such type of combinatorial optimization tasks. NNs have been already explored for solving the TSP for several decades. One of the earliest NN models for the TSP was presented by Hopfield and Tank in 1985 [12], the Hopfield Neural Network (HNN), later improved in [13]. The HNN is a fully connected recurrent NN where the nodes are binary units updated synchronously based on a specific activation function. Then, a group of self-organizing structures such as Self-Organizing Map (SOM) [14], FLEXMAP [15], or relatively recent Growing Self-Organizing Array (GSOA) [16] have been proposed. These models used unsupervised learning to discover a structure in the input data and can find solutions of the TSP that are competitive to existing traditional heuristics.

More recently, NNs have been applied to the TSP using a Convolutional Neural Network (CNN) [17], Attention Model (AM) [18, 19], and Graph Neural Network (GNN) [20]. These learning-based models have achieved breakthrough results in computer vision, text processing, and speech recognition. However, their effectiveness for the TSP has been limited compared to traditional heuristic-based methods. Despite that, combining NNs with heuristics [21, 22] has been shown promising to improve TSP solutions.

Moreover, a generalization of the TSP into multi-vehicle formulation of the Vehicle Routing Problem (VRP) [23] has applications in logistics, transportation, telecommunications, or urban planning. For example, in logistics, the VRP is used to find optimal routes for vehicles and drivers to make deliveries or goods pick up from multiple locations. By solving instances of the

VRP, logistics companies can reduce transportation costs, improve delivery times, and increase customer satisfaction. The VRPs can be used to optimize the routing of public transport vehicles, such as buses and trains, technicians and repair crews in telecommunications, or even a waste collection vehicle fleet. Every day, enormous amounts of data are solved using VRP algorithms, resulting in a vast amount of solved instances. However, these instances are rarely reused due to differences in problem parameters or constraints between instances. Therefore, NNs are a promising approach for reusing already solved instances since they already exhibited excellent results in abstracting data and identifying patterns. By training neural networks on solved instances, they can learn to generalize to new instances with different parameters or constraints. Additionally, NNs can be used to predict similar solutions for unsolved instances, helping to guide heuristic-based algorithms to improve the solution quality more quickly. Such ideas are the primary motivations for the research topic addressed in the presented thesis. In particular, the aim of the thesis is as follows.

The thesis aims to explore the potential of reusing already solved TSP instances by utilizing existing NN techniques combined with powerful combinatorial metaheuristic(s). The goal is to find how well the NN can perform in solving the TSP and how it can be utilized in a new hybrid approach. The proposed method combines the NN and similarity search algorithm with a heuristic algorithm. The NN is trained on a large dataset of TSP instances and then used to provide input for the similarity search algorithm that finds similar instances to the current instance. Then, it reuses the solution of the most similar instance to create the initial solution for a heuristic method. Such an initial solution is then improved by the combinatorial metaheuristic within a given computational time. The central hypothesis of the thesis is the expectation that a solution determined from an available solution of a similar instance would yield a better solution than without the similarity matching. Moreover, an initial solution would yield an improved solution in less computational time, or a better solution is determined within the same computational time for an improving heuristic.

In the following chapter, the NN-based approaches are presented. Then, the TSP and the idea of the problem approach are introduced in Chapter 3. The utilized existing approaches for the hybrid approach are presented in Chapter 4 followed by Chapter 5 describing the proposed hybrid method which is evaluated in Chapter 6.



# Related Work

One of the first Neural Network (NN)-based approaches (if not the first) to the Traveling Salesmen Problem (TSP) has been proposed in the 80s by Hopfield and Tank [12], and the learning network model is called Hopfield Neural Network (HNN). In the HNN model, neurons represent the locations of the TSP, and the neurons' connections are the distances. The HNN minimizes the total distance by adjusting the weights between neurons to minimize the so-called energy function. Although the first implementation by Hopfield and Tank was criticized, modifications of the energy function were deemed necessary by other researchers. The feasibility was guaranteed by a formulation presented by Aiyer, Niranjan, and Fallside [13].

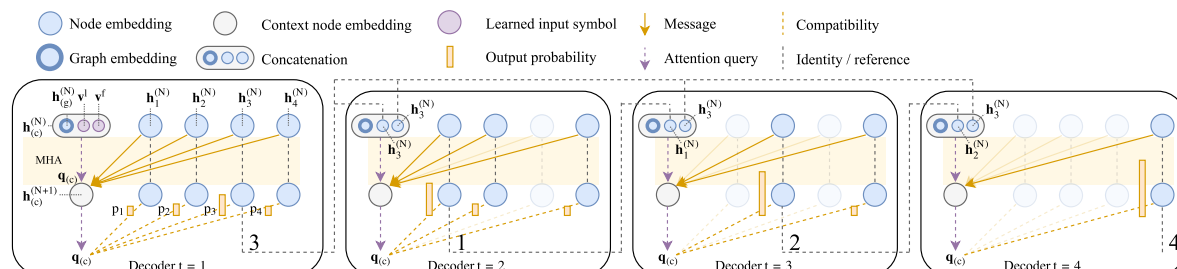
Neurons representing locations are used in Self-Organizing Map (SOM)-based solvers [24]. The SOM has been originally proposed as a technique to project high-dimensional data into a 2D lattice suitable for visualization; therefore, neurons are placed in a plane. The first application of SOM to the Euclidean instances of the TSP has been proposed by two independently introduced approaches by Angéniol *et al.* [14] and Fort [25] in 1988. The neurons connected into a ring represent a tour that acts like an elastic structure with similar behavior to the elastic net [26], which uses a different learning mechanism. Based on the ring concept, the FLEXMAP was proposed by Fritzke and Wilke [15] in 1991. Besides, several improvements of SOM-based solvers have been proposed, such as [27], and generalization to Non-Euclidean instances motivated by robotics inspection tasks [28]. Further, the unsupervised learning of SOM evolved into Growing Self-Organizing Array (GSOA) [16], where the ring grows and shrinks dynamically based on the size of the TSP instance, which represents a flexible heuristic approach to various routing problems [29].

In addition to the unsupervised learning-based methods, supervised learning-based Machine Learning (ML) approaches to the TSP are particularly interesting within the context of the thesis. Therefore, the contemporary approaches Convolutional Neural Network (CNN), Attention Model (AM), and Graph Neural Network (GNN) are presented in Sections 2.1 to 2.3, respectively. Then, hybrid approaches combining learning and non-learning methods are introduced in Sections 2.4 and 2.5.



## 2.2 Attention Model

Another learning-based model applied to the TSP is based on a fundamental element of the transformer architecture [30], the attention layer. A representative of attention layer-based approaches to the TSP is the Attention Model (AM) presented in [18] that address some of the drawbacks of the CNNs. It works as follows.



**Figure 2.2:** The attention-based decoder is based on the probability of being the next path node successor. The process is repeated without already selected nodes until all nodes have been subscribed to a path order. The image is adopted from [18].

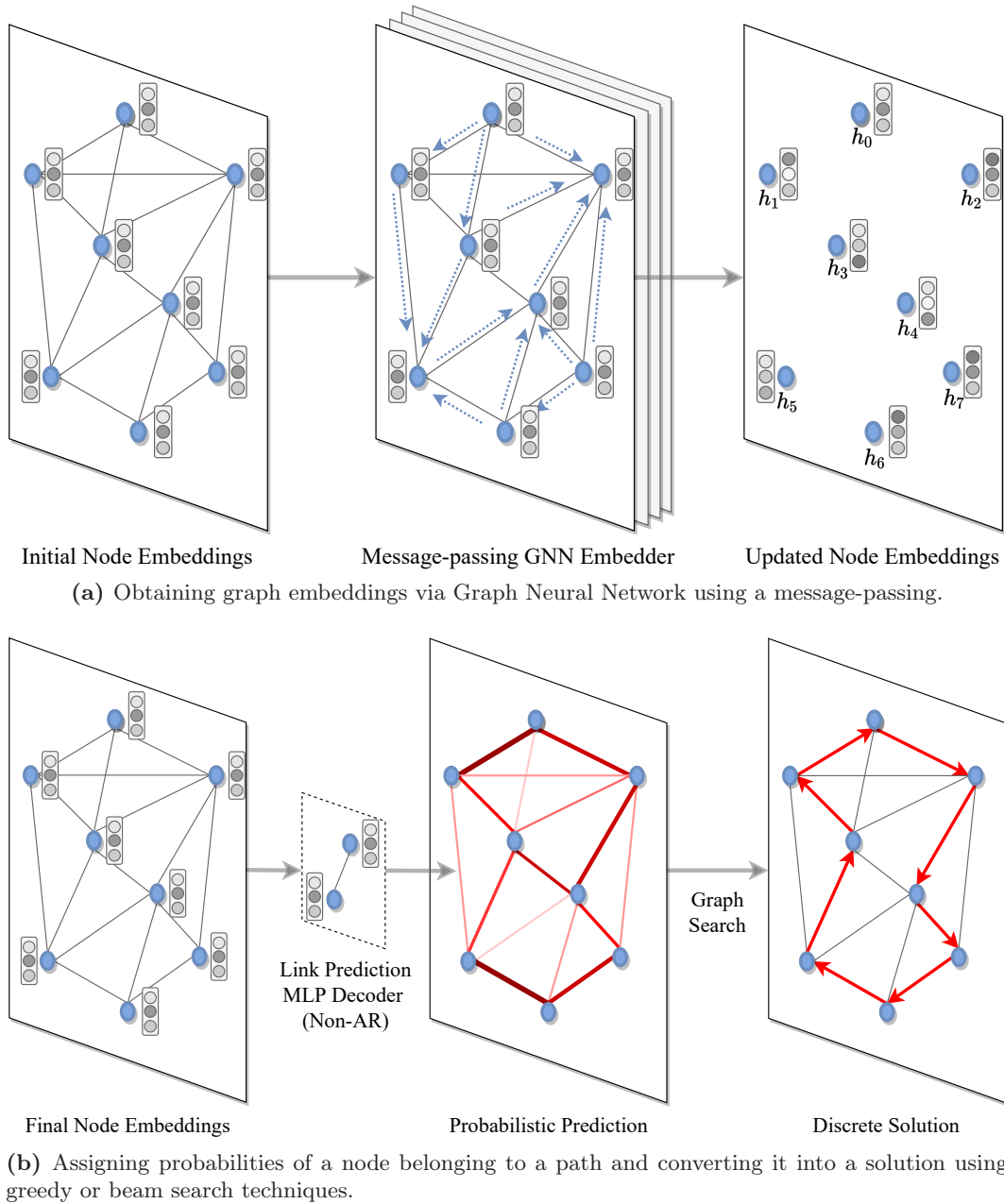
An attention layer is a part of the Neural Network (NN) processing pipeline that computes a weighted average of a set of vectors, where the weights are determined based on the similarity between each vector and a context vector, which can be computed by the layer itself or provided as an input. The returned node embeddings are invariant to the input order; unlike the transformer model, it does not use a positional encoding. The AM decoder, depicted in Fig. 2.2, computes a probability distribution over the neighbor nodes of the current node, indicating which node is most likely to be selected as the path node successor. The node with the highest probability is then selected as the next node in the solution sequence and masked out. The process is repeated until all nodes have been selected.

The reported results of the AM are computed with the optimal solution provided by the Gurobi optimizer and Concorde. Besides, heuristic solutions are examined using LKH [31] and simple heuristics that include nearest insertion, random insertion, farthest insertion, and nearest neighbor. The AM provides solutions faster than the optimal solvers and LKH. The solutions found are significantly better than simple heuristics with competitive computational time. However, the model is trained only on randomized data generated by a normal distribution in a unit square. Moreover, the AM was trained with a fixed number of nodes in the instances. Provided models for 20, 50, and 100 nodes were only studied on the same size instances.

*Performed empirical validation of the model* – Based on our evaluation, the method does not generalize on the TSPLIB [32], see aggregated results reported in Fig. 6.2 and the full results in Tables A.1 to A.4. Nevertheless, we consider AM in the proposed solution to the TSP.

## 2.3 Graph Neural Network

A relatively recent model with reported promising results is presented in [20] that is based on the Graph Neural Network (GNN) [20]. It accepts an instance of the TSP as a graph representation. Using an encoder of the GNN, nodes collect information from their neighbors



**Figure 2.3:** An overview of the encoder-decoder of the Graph Neural Network (GNN) for the TSP; (a) the encoder providing node embeddings and; (b) the decoder based on searching methods. The figures are adopted from [20].

via recursive message passing. Thus, a local graph representation, as node embeddings, is created, see Fig. 2.3a. However, for large TSP instances, the pairwise computation required for message passing can be demanding. Therefore, a sparsification process based on k-nearest neighbors can be employed to reduce the computational load.

The GNN decoder depicted in Fig. 2.3b assigns probabilities to each node that is a part of the solution. It can be achieved through two methods: non-autoregressive decoding and autoregressive decoding. In non-autoregressive decoding, the probabilities are assigned in-



independently to each edge in the graph without considering their sequential ordering. The probabilities are converted into discrete decisions using a regular graph search technique, such as greedy or beam search. In contrast, autoregressive decoding explicitly models the sequential inductive bias of a TSP tour through step-by-step graph traversal. It is done using an attention-based decoder, as in Section 2.2, that outputs a probability distribution over the neighbors of a node at each step.

The GNN has been empirically assessed only on relatively small instances with 20, 50, 100, and 200 locations, which optimal solvers can solve quickly. Furthermore, the instances were randomly generated the same way as for the AM described in Section 2.2. Additionally, the authors examined the generalization of the model trained on instances with variable sizes from 20 to 50 locations. However, the solution quality is reported only with a comparison to the optimal solver Concorde and simple furthest insertion heuristic. The authors further study several factors in the randomized data, such as generalization to large instances. The reported optimality gap increases rapidly for larger instances than it was trained on. Surprisingly, the model trained on instances with 100 locations generalized on larger instances better than the model learned on instances with 200 locations.

*Performed empirical validation of the model* – Based on the promising results reported in [20], we assessed the provided models ourselves. From our results listed in Fig. 6.2 and in Tables A.1 to A.4, two GNN models trained on instances with 20–50 locations and instances with 200 locations (further denoted as the GNN 20-50 and GNN 200) provide worse results than the AM model. The GNN 200 cannot compete even with the greedy algorithm. Therefore, we do not consider GNN in the proposed solution to the TSP.

## 2.4 NeuroLKH

The NeuroLKH [22] is a hybrid algorithm that combines NNs and the heuristic LKH for solving the TSP. It uses a Sparse Graph Network (SGN) trained with supervised and unsupervised learning to assign scores to edges and penalties to nodes. These scores and penalties improve the performance of solely used LKH. NeuroLKH uses the output of the SGN to generate an edge candidate set and to guide the search process of the LKH by transforming edge distances. See a visualization of the NeuroLKH depicted in Fig. 2.4.

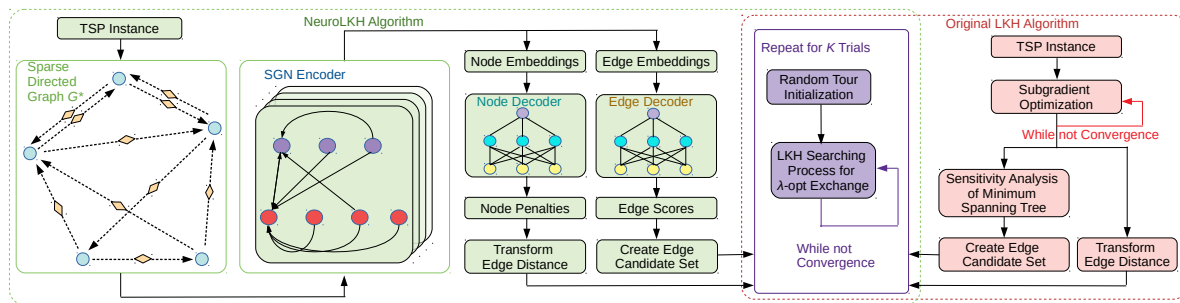


Figure 2.4: A visualization of the NeuroLKH adopted from [22].

The reported results for the NeuroLKH are based on the trained NeuroLKH randomized datasets only but compared with the LKH [11]. The NeuroLKH managed to obtain improved results within the same computational time compared to the LKH with a slight difference from the optima. Combining the NN method and LKH allows the NeuroLKH to search the solution space more efficiently.



Although the reported results of the NeuroLKH are promising, it does not provide encoding that can be used straightforwardly as a feature vector for an instance descriptor. Therefore, we do not consider it in the proposed solution.

## 2.5 Neural Large Neighbourhood Search

The Neural Large Neighborhood Search (**NLNS**) is a combination of the traditional combinatorial heuristic Large Neighborhood Search (**LNS**) with deep learning [21]. The idea is to determine an initial solution to which the **LNS** explores the search space and improves the solution. The **LNS** is a combinatorial metaheuristic technique that systematically explores large sets of candidates. The proposed idea is to employ **NN** to suggest promising neighborhoods to examine.

Regarding the combination of **NN** with existing a regular combinatorial heuristic, the **NLNS** represents a powerful approach for solving combinatorial optimization problems. It was proposed for the Capacitated **VRP** and the Split Delivery **VRP**. In both problems, the **NLNS** outperforms the **AM** with sampling [18] and reinforcement learning approach with beam search [33] in the solution quality and also the required computational time. Moreover, the results of the **NLNS** are compared to the **LNS**, **LKH** [31] and Unified Hybrid Genetic Search (**UHGS**) [34, 35]. The **NLNS** provides better solutions than the non-learning **LNS** within less computational time. On the other hand, it gives competitive results to the **LKH**, but the **UHGS** outperforms all compared methods in the solution quality and time demands. The **NLNS** leverages the strengths of the learning and non-learning approaches to generate high-quality solutions quickly and efficiently. Nevertheless, we do not consider the **NLNS** in the proposed solution because it is not optimized to the regular **TSP**.

## 2.6 Conclusion on Solving the TSP by Neural Networks

The presented selected **NN**-based approaches to the **TSP** suggest that standalone **NN** models are insufficient to provide high-quality solutions competitive to the well-established combinatorial heuristics such as the **LKH**. On the other hand, combining **NN**-based techniques with the regular approaches to the **TSP** can leverage their advantages. **NN**-based models excel in their adaptability to pre-solved solutions, while regular approaches demonstrate effectiveness in exploring the combinatorial space of feasible solutions. By integrating the approaches, we expect to leverage the strengths of both paradigms and achieve improved performance and results. It is the central idea and the hypothesis is answered in the thesis.

# Problem Statement

In the thesis, we study the Traveling Salesmen Problem (TSP) intending to employ recent advancements on NN-based approaches combined with existing powerful heuristic solutions in a hybrid approach that would increase the quality of found solutions and computational requirements. We aim to develop a method to quickly find, possibly high-quality, initial solutions that a combinatorial metaheuristic can improve further. Further, we aim to answer whether we can improve the solution quality using optimal (or high-quality) solutions for some instances that can be similar to a new instance. Therefore, in the rest of the chapter, the TSP is formulated and formally defined in Section 3.1, and an overall idea of the expected hybrid approach is introduced in Section 3.2.

## 3.1 Euclidean Traveling Salesman Problem

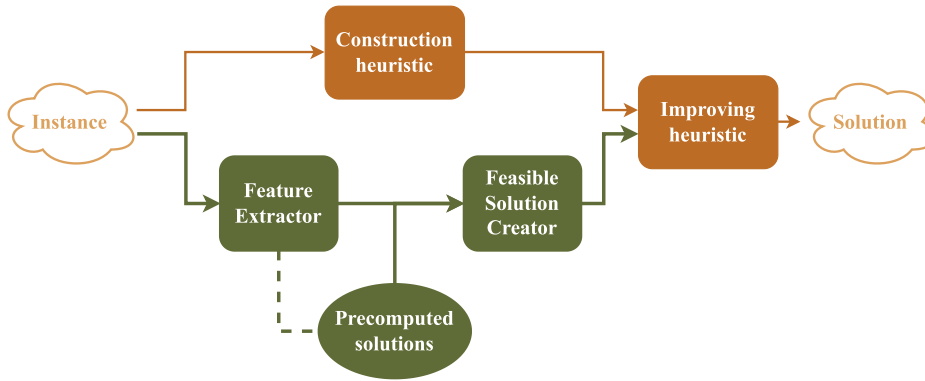
Due to the almost exclusive focus of existing NN-based methods on Euclidean instances of the TSP in a plane, we consider Euclidean TSP in the rest of the thesis, which can be formally defined as follows. Let  $V = \{\mathbf{v}_1, \dots, \mathbf{v}_n\}$  be a set of two-dimensional locations to be visited and  $\|\mathbf{v}_a - \mathbf{v}_b\|$  denotes the Euclidean distance between any two locations  $\mathbf{v}_a$  and  $\mathbf{v}_b$ . The TSP can be formulated as an optimization problem to find an optimal sequence  $\Sigma$  of visits to the  $n$  given locations such that the length of the closed path (denoted  $\mathcal{L}(\Sigma)$ ) visiting the locations and returning to the initial location is minimized. The solution can be represented as a sequence of location indexes  $\Sigma = \langle \sigma_1, \dots, \sigma_n \rangle$ . The problem is formulated and summarized in Problem 3.1.

**Problem 3.1 Traveling Salesmen Problem (TSP).**

$$\begin{aligned}
 \min_{k, \Sigma} \quad & \mathcal{L}(\Sigma) = \|\mathbf{v}_{\sigma_1} - \mathbf{v}_{\sigma_n}\| + \sum_{i=2}^n \|\mathbf{v}_{\sigma_{i-1}} - \mathbf{v}_{\sigma_i}\| \\
 \text{s. t.} \quad & \Sigma = \langle \sigma_1, \dots, \sigma_n \rangle \\
 & \sigma_i \in \{1, \dots, n\} \\
 & \forall i, j, i \neq j : \sigma_i \neq \sigma_j
 \end{aligned} \tag{3.1}$$

## 3.2 Hybrid Approach

The hybrid approach combines some construction heuristics with an improvement procedure to find a better solution than the initial solution provided by the construction heuristic. We plan to employ existing NN-based methods as a construction heuristic to quickly find some initial feasible solution to a particular instance of the TSP. Nevertheless, we can also use any traditional construction heuristic. However, we also aim to employ a set of precomputed solutions and determine a suitable technique to represent an instance of the TSP to retrieve a similar instance with an available solution. Then, we like to use the solution of a similar instance to construct a solution of a new instance that can be further improved by the improving heuristic. The pipeline to validate the hypothesis that solving the TSP can benefit from available solutions of similar problems is depicted in Fig. 3.1.



**Figure 3.1:** The proposed idea of the hybrid pipeline is to solve the TSP using an alternative to regular construction heuristics followed by an improving heuristic. The orange pipeline represents approaches using regular heuristics. The green pipeline represents the proposed approach of obtaining an initial solution using a set of available solutions using a similarity matching of the instances.

A solution pipeline using regular (traditional) combinatorial heuristics with a constructive phase generating the initial solution followed by an improving phase is depicted in orange in Fig. 3.1. The construction heuristic can be a simple method such as random or cheapest insertion. Then, the improving heuristic has to make many steps to obtain a decent solution. Thus, the initialization based on already solved solutions could help decrease the number of steps in the improving phase or even enhance the result. However, the problem is determining a similar instance to the given one. Here, we propose to utilize the NN-based approach to provide a data abstraction over the input data. Then, based on the similarities within the set of already solved instances, we can retrieve a solution of the most similar instance and use it to create a feasible solution. The proposed pipeline is depicted in green in Fig. 3.1.

### 3.2.1 Research Question

Within the hybrid approach, we aim to answer the following research question.

- Would usage of the NN-based instance characterization yield a similarity measure to retrieve a similar instance with a solution that would yield an improved initial solution than a traditional construction heuristic?

# Background

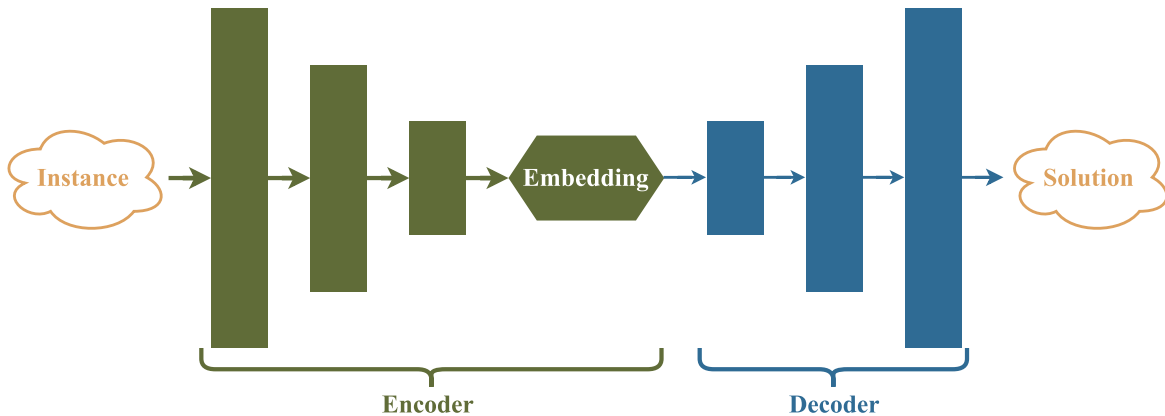
The proposed approach to combine NN-based methods with existing techniques to the TSP is to find an alternative to construction heuristics that will provide a better solution (best in less computational time) and then improved by employing an improving heuristic. Based on the literature review on NN-based methods, we aim to develop a solution pipeline that would better scale with the instance size.

We target solving instances with hundreds of locations, while the reported learned-model are trained on tens up to 100 locations. Therefore, we propose to exploit a set of available solutions to construct a new instance solution using an existing solution for the most similar instance. However, finding a similar instance might be tricky using the standard properties of a graph. Therefore, we propose to utilize embeddings of the NN-based methods to extract a feature vector characterizing the instance. We assume that NN can learn different characteristics of high dimensional input data, and only by activating the most important neurons creates a “non-human,” low-dimensional description of the input data that can be used for similarity assessment.

We follow the proposed pipeline depicted in Fig. 3.1 and consider existing techniques described in the rest of this chapter as expected building blocks for the development of the pipeline. In particular, we consider encoder-decoder architecture, and embedding as suitable techniques to address the similarity assessment of a new instance and instances with known solutions. Moreover, we consider GSOA [16] as an unsupervised learning-based technique to the TSP that can initialize a solution to different instances. Therefore, we provide a brief overview of GSOA in Section 4.2 to make the thesis self-contained. Further, our choice for an improving heuristic is the combinatorial metaheuristic Variable Neighborhood Search (VNS) because it has already been successfully deployed to several routing problems at the training workplace, where the thesis is solved. The fundamental principles of the VNS are overviewed in Section 4.3.

## 4.1 Encoder-Decoder Architecture

The encoder-decoder architecture is a NN architecture commonly used for image segmentation, machine translation, speech recognition, and other sequence-to-sequence tasks where the



**Figure 4.1:** A principled schema of the encoder-decoder architecture. The green part on the left encodes the input data and provides embeddings. Then the blue decoder on the right yield a result base on the embedding.

TSP might be seen as a graph-to-graph translation. The basic idea of the encoder-decoder architecture is to use two separate neural networks. In Fig. 4.1, the encoder network is drawn in green, and the decoder network is depicted in blue. The encoder network takes in an input, such as an image, a sequence of words, or a graph, and converts it into a fixed-size representation, called the embedding, which contains all the relevant information needed to perform the task. The decoder network takes the bottleneck representation generated by the encoder and generates an output.

An embedding is a vector representation of a high-dimensional data point in a lower-dimensional space. It is learned through a NN which maps the input data into a lower-dimensional representation that preserves the most relevant features of the data. Such a lower-dimensional representation can be used as a feature vector in downstream tasks. We aim to utilize it as a descriptor to determine a similarity score of two TSP instances.

### Embedding as a Feature Vector

A feature vector is a mathematical representation describing a particular object or phenomenon. It is a list of numerical values that represent various characteristics or features of data. In the context of signal processing, including images, where NN-based methods exhibited great results, features can be color, shape, size, texture, or other properties of an object, such as various acoustic or linguistic features of a sound or language. Feature vectors are commonly used in machine learning and pattern recognition algorithms, where they are used to represent the input data that the algorithm uses to learn or classify patterns.

Particular existing examples are feature vectors representing the intensity and color of each pixel in an image. In speech recognition, a feature vector can represent the frequency and amplitude of each sound wave. In natural language processing, word embeddings are used to represent words as feature vectors. These embeddings are learned through a neural network that predicts the context of a word in a text corpus based on its surrounding words. The resulting word embeddings capture the semantic and syntactic similarities between words and can be used to perform sentiment analysis, text classification, or machine translation. Similarly, in computer vision, image embeddings can be learned through a CNN that extracts features from the image and maps them to a lower-dimensional space. These embeddings can

be used as feature vectors for object detection, image classification, or image retrieval tasks.

We assume similar properties can be achieved for TSP instances, where an embedding of the learned NN can be used as the feature vector (descriptor) for computing the similarity score of an instance with instances from a Set of Solutions (SoS) with known solutions. Various techniques can be used to compute the similarity score, including Kullback–Leibler divergence for probability distributions. However, in the present work, we consider L2 Norm and the nearest neighborhood algorithm to determine the most similar instances from the SoS.

## 4.2 Growing Self-Organizing Array

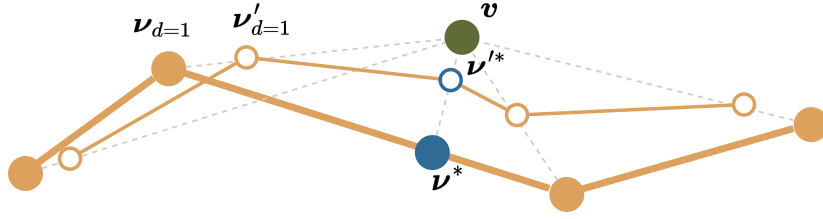
The Growing Self-Organizing Array (GSOA) [16] is an unsupervised learning-based heuristic that originates from SOM. Even though it is an iterative procedure, it can be considered a construction heuristic because once the model is stabilized, the solution is no longer improving with increasing learning epochs. The procedure iteratively adapts an array of nodes representing a path (ring) in the input space. The nodes are connected by straight-line segments into which a new node can be added during the learning. In each learning epoch, all locations to be visited are examined randomly to avoid local minima. For each location, the best matching unit is determined using the shortest distance between the location and segments of the ring. The segment’s point closest to the location is used to insert a new node into the ring, and the node is denoted winner for the particular location. Then, the winner node and its neighboring nodes are adapted toward the locations with the power defined by a learning function depicted in Fig. 4.2. Since a new node is added for each location in every learning epoch, all nodes from the previous epoch are removed from the array (ring) at the end of each epoch. Hence, the array contains the same number of nodes as the number of locations. Furthermore, each node has an associated location, and thus the sequence of visits to all locations can be retrieved by traversing the array.

The adaptation is a de facto movement of the node position  $\boldsymbol{\nu}$  toward the location  $\boldsymbol{v}$  according to the neighboring function

$$\boldsymbol{\nu}' = \boldsymbol{\nu} + \mu e^{-d^2/G^2}(\boldsymbol{v} - \boldsymbol{\nu}). \quad (4.1)$$

The power of adaptation is decreased with the distance  $d$  of the neighboring node  $\boldsymbol{\nu}$  from the winner node  $\boldsymbol{\nu}^*$ . Besides, the power is further controlled by the learning rate  $\mu$  and learning gain  $G$ , decreased using the gain-decreasing rate at the end of each learning epoch. Thus, depending on the initial value of the learning gain and the gain decreasing rate, after a specific number of epochs, the gain becomes so low that no adaptation is performed, and the array (ring) becomes stable. Therefore, the GSOA runs in a fixed number of epochs. An interested reader is referred to [16] for a detailed description of the adaptation, convergence, and parameter settings.

The array of nodes is adapted in the input space; therefore, the array can be straightforwardly initialized from some, even partial, solution. A solution as a sequence of locations can be directly used as the initial array (ring) of nodes. Depending on the value of the initial learning gain, the power of the adaptation might completely change the initial solution (high values), or only small adjustments can be made.



**Figure 4.2:** The GSOA ring adaptation where each node  $\nu$  is dragged toward the green location  $v$  based on distance from the winner node  $\nu^*$ . The filled orange nodes represent the ring state before adaptation and the new state is depicted by the outlined nodes.

### 4.3 Variable Neighborhood Search

The Variable Neighborhood Search (VNS) [7] is a general metaheuristic framework for solving combinatorial optimization problems. The VNS systematically searches through the neighborhood space of feasible solutions using predetermined neighborhood operators. Within the context of the TSP, the neighborhood can be defined as all possible sequences of visits to the locations created by applying a particular operator, such as exchanging two nodes in the sequence. The VNS algorithm starts with an initial solution (e.g., found by a construction heuristic) and iteratively explores the solution space by using Shake and Local Search procedures. The Shake procedure is applied to the current solution to escape local optima, while the Local Search iteratively applies different operators to improve a candidate solution further. If the solution is improved, it is accepted as the current best solution found, and the search continues. The algorithm is summarized in Algorithm 1.

---

**Algorithm 1:** Variable Neighborhood Search (VNS)

---

**Input:**  $V = \{v_1, \dots, v_n\}$  –  $n$  locations to be visited,  $\Sigma_{\text{init}} = \langle \sigma_1, \dots, \sigma_n \rangle$  – initial sequence of visits to the set of  $V$ .

**Output:**  $\Sigma$  – The sequence of visits to the set of  $V$ .

---

```

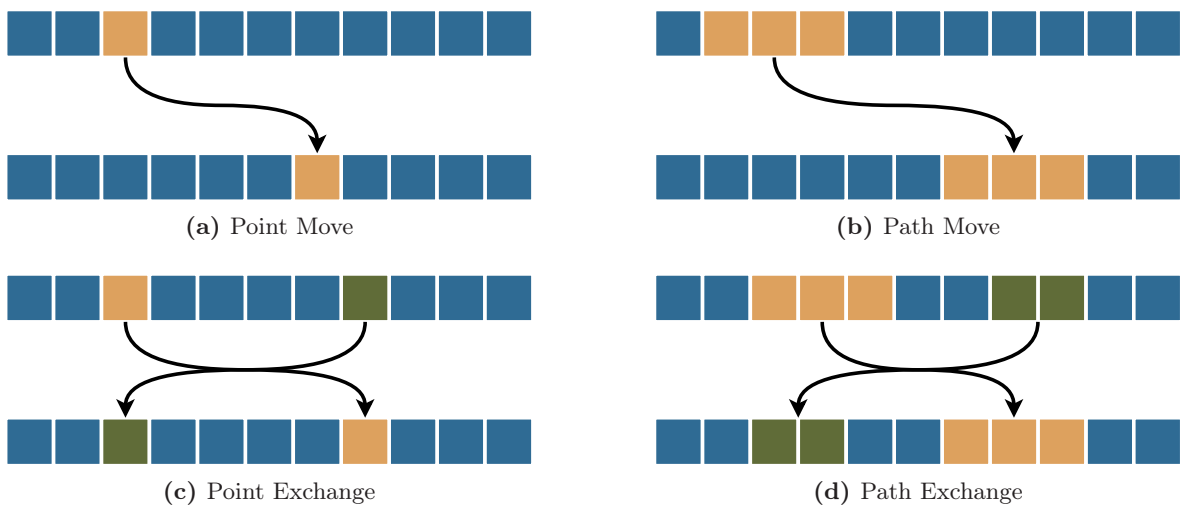
1  $\Sigma \leftarrow \Sigma_{\text{init}}$ 
2 for stopping condition is not met do
3    $\Sigma' \leftarrow \text{shake}(V, \Sigma)$ 
4    $\Sigma'' \leftarrow \text{localSearch}(V, \Sigma')$ 
5   if  $\mathcal{L}(\Sigma'') < \mathcal{L}(\Sigma)$  then
6      $\Sigma \leftarrow \Sigma''$ 
7   end
8 end
9 return  $\Sigma$ 

```

---

In the regular VNS, the neighborhood operators are applied systematically, which can be too demanding. Therefore, the Randomized Variable Neighborhood Search (RVNS) variant randomly searches the neighborhood space up to  $n^2$  operations, where  $n$  is the instance size. Furthermore, the performance of the VNS highly depends on the operators' demands. In [36], the authors use four operators depicted in Fig. 4.3.

The Shake procedure uses the Path Move operator, which shifts a path subsequence to



**Figure 4.3:** Four Variable Neighborhood Search operators used in [36].

a different position, and the Path Exchange operator, which picks two non-overlapping path parts and exchanges them. The Local Search uses the Single Point Move and the Single Point Exchange. These operators are also used to employ the RVNS as the improving heuristic in Fig. 3.1.





# Proposed Hybrid Approach

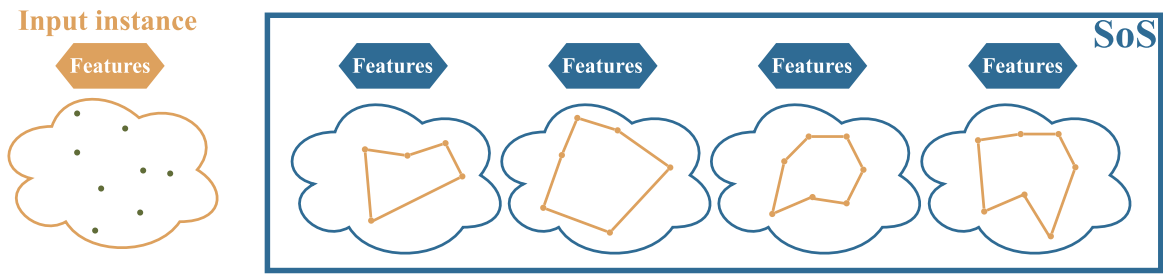
Following the proposed architecture in Fig. 3.1, we develop a novel method to utilize Neural Network (NN) methods for addressing the re-usage of already solved instances. Several pipelines can be implemented based on a specific implementation of the particular blocks in Fig. 3.1. However, our main focus is on the Feature Extractor to access the set of pre-computed solutions and the Feasible Solution Creator (FSC), described in the following parts of this chapter. Moreover, we detail the used improving heuristic in Section 5.4, based on the RVNS.

## 5.1 Set of Solutions Concept

The idea of the Set of Solutions (SoS) is motivated by the assumption that practical instances might be similar in overall shape but can change in detail. Specifically for large instances in logistics, we can imagine that major cities to be visited might be a few, but changes can be in the particular locations within the city area. Therefore, we propose to create a SoS where we can “index” an instance by a similarity measure using an instance descriptor.

Although clustering and other relatively intuitive methods can be utilized for the Euclidean TSP, we are also motivated by further extensions towards non-Euclidean instances and possibly the VRP generalized to time windows where such intuition does not work. Therefore, we propose to utilize an instance descriptor that can provide an abstraction over high-dimensional data. Then, the extracted instance descriptor can be matched with instances from the SoS to find a solution for a similar instance.

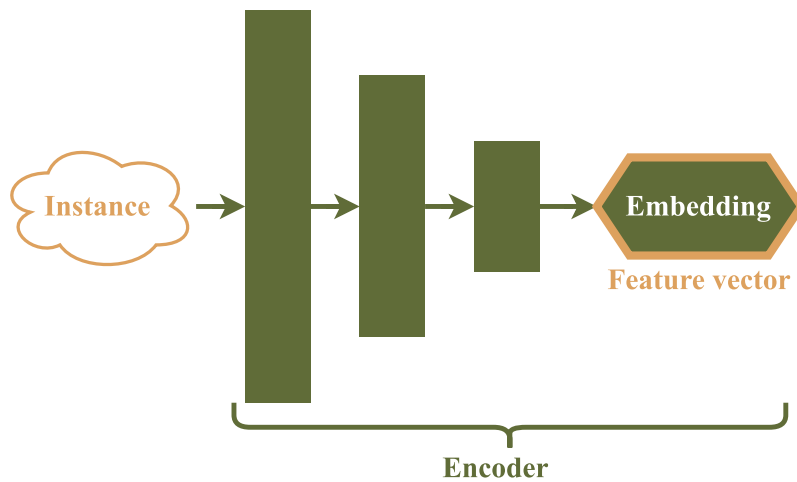
Let us assume, we have a feature vector characterizing an input instance shown in Fig. 5.1. The similarity can be measured based on the L2 Norm between the feature vectors of the query instance and instances in the SoS. By finding the instance descriptor with the smallest L2 Norm, the most similar instance to the query instance is obtained. Then, the solution of the query instance can be based on the solution of the most similar instance from the SoS. The particular feature descriptor is described in the following section.



**Figure 5.1:** Concept of the Set of Solutions (SoS) and input instance. By retrieving the most similar instance from the SoS to the input instance based on the smallest distance of the feature vectors, a solution with some resemblance to the input instance is obtained.

## 5.2 Feature Extractor

We propose to use NN-based feature extractor and the corresponding feature descriptor. The motivation is to utilize a part of some sequence-to-sequence NN for the TSP. We assume that the bottleneck information in the encoder-decoder architecture, alias the embedding, should provide enough information to retrieve the solution in the decoder part. Thus, the encoder separately, depicted in Fig. 5.2, can be used as a feature extractor, and its provided embedding can be seen as the feature descriptor.



**Figure 5.2:** Neural network encoder used as a feature extractor. The NN encoder takes a TSP instance as the input and outputs embedding that is directly used as an instance descriptor.

The particular NN approach is the encoder-decoder architecture of the Attention Model (AM) presented in [18] because of reported promising results of the TSP. The authors of [18] made three pre-trained models for the TSP available, which can be directly used. Hence, a pre-trained model taught on the instances with 100 locations has been deployed in the proposed solver due to the best generalization to larger instances and the empirical results are reported in the following chapter. However, it is necessary to address possible different scales of the TSP instances, and therefore, the instances need to be normalized.

## Instance Space Normalization

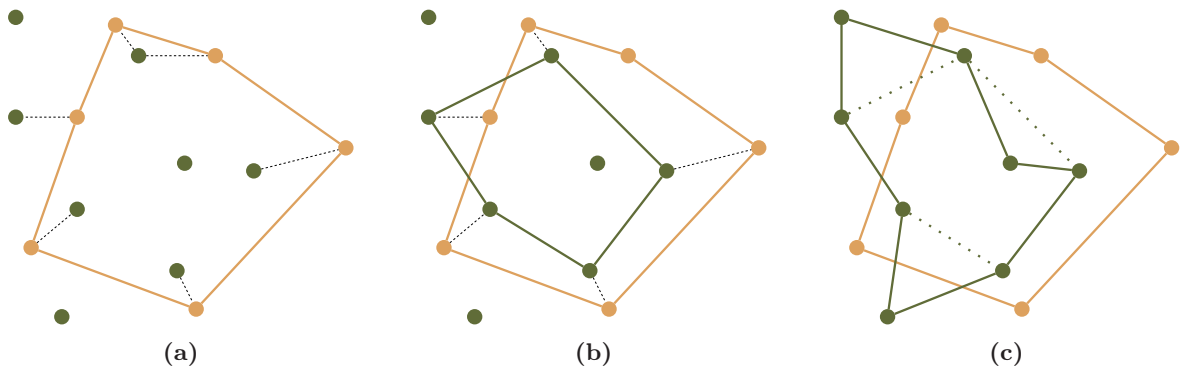
The TSP instances can highly vary, however, the instance space depends on the Feature Extractor method. Since the used extractor has been trained on random instances generated in a unit square, it is necessary to scale any instance to a similar space. Thus, the input locations  $V = \{\mathbf{v}_1, \dots, \mathbf{v}_n\}$  need to be scaled into a unit square. Furthermore, the locations are shifted to fill the unit square evenly within the intervals from zero to one;  $\langle 0, 1 \rangle \times \langle 0, 1 \rangle$ .

## 5.3 Feasible Solution Creator

Having a solution of a similar instance from the SoS to the query instance, the solution has to be mapped to the input instance and create a feasible solution. We propose two approaches, a greedy method, and employment of GSOA, that are detailed in Section 5.3.1 and Section 5.3.2, respectively. Note that the query instance is normalized, and the solution is found in the normalized input space. Once a sequence of visits is determined, the original instance is utilized to determine the solution length of the original input instance.

### 5.3.1 Greedy Construction of Feasible Solution

The greedy feasible solution construction is based on pairing each location in the solution with the location of the input instance. Then, if the size of the input instance is larger than the similar instance, the locations are added to the initial tour using *cheapest insertion*. An example of the construction is visualized in Fig. 5.3, and it works as follows.



**Figure 5.3:** The greedy FSC first chooses the closest locations of the input instance to a similar instance. If the input instance location has matched more than one similar instance location, the closest similar location is paired. The remaining locations are added to the path using the cheapest insertion.

Since the solution is a sequence of indices, the pairing is based on the shortest distances between the locations so that the similar solution's sequence indices are assigned to the closest input instance locations. Thus, a partial solution is determined as depicted in Fig. 5.3b. Then, the remaining locations not visited by the partial solution are greedily inserted between two locations such that the prolongation of the closed path is minimal, see Fig. 5.3c.

The greedy feasible solution construction is relatively straightforward and does not consider possible rotation changes of the input instance and the retrieved similar instance from the SoS. Therefore, we further consider feasible solution construction using GSOA.

### 5.3.2 GSOA-based Construction of Feasible Solution

Mapping a different instance solution to the input instance is straightforward with GSOA. Each node in the array of nodes corresponds to a location in the input space. Thus, the initialization of the array is direct usage of the similar instance solution. However, we might need to adjust the initial learning gain to prevent considerable adaptation from the initial solution. Therefore, the suggested learning gain 10 [16] is lowered to 5. The value of the learning gain has been empirically determined as the lowest value without a significant impact on the solution quality. Since the learning gain directly affects the number of learning epochs, lower values decrease the computational time. All the other parameters of GSOA are used as suggested in [16].

## 5.4 Improvement Heuristic

The RVNS is employed as the improving heuristic with the operators used in [36]. The implementation consists of the **shake** procedure, modifying the current solution. It randomly chooses between two operators

1. **pathMove**( $\Sigma$ ):

$$\begin{aligned} \sigma_i &= \sigma'_{(i+r \bmod n)+1} \quad \text{for } 1 \leq r \leq n, \\ \Sigma' &= \langle \sigma'_1, \dots, \sigma'_{a-1}, \sigma'_{b+1}, \dots, \sigma'_c, \sigma'_a, \dots, \sigma'_b \sigma'_{c+1}, \dots, \sigma'_n \rangle \\ &\quad \text{for } 1 \leq a \leq b \leq n, c \in \{1, \dots, a-1, b+1, \dots, n\} \end{aligned} \quad (5.1)$$

2. **pathExchange**( $\Sigma$ ):

$$\begin{aligned} \sigma_i &= \sigma'_{(i+r \bmod n)+1} \quad \text{for } 1 \leq r \leq n \\ \Sigma' &= \langle \sigma'_1, \dots, \sigma'_{a-1}, \sigma'_c, \dots, \sigma'_d, \sigma'_{b+1}, \dots, \sigma'_{c-1}, \sigma'_a, \dots, \sigma'_b, \sigma'_{d+1}, \dots, \sigma'_n \rangle \\ &\quad \text{for } 1 \leq a \leq b < c \leq d \leq n \end{aligned} \quad (5.2)$$

Both operators shift the path indexed by  $r$  to new  $\sigma'_i$  to obtain an even subpath going over the start and end of the sequence  $\Sigma$  because the TSP solution is a closed tour. The operator **pathMove** randomly chooses  $a$ ,  $b$ , and  $c$  indices of the sequence and puts the sequence  $\langle \sigma'_a, \dots, \sigma'_b \rangle$  after the location index  $\sigma'_c$ , which is depicted in Fig. 4.3b. Whereas **pathExchange** switches two non-overlapping subsequences defined by randomly chosen  $a$ ,  $b$ ,  $c$ , and  $d$ .

The Local Search iteratively explores the solution space by the same operators used in the **shake** procedure but with two additional operators:

3. **pointMove**( $\Sigma$ ):

$$\begin{aligned} \Sigma' &= \langle \sigma_1, \dots, \sigma_{a-1}, \sigma_{a+1}, \dots, \sigma_b, \sigma_a, \sigma_{b+1}, \dots, \sigma_n \rangle \\ &\quad \text{for } 1 \leq a \leq n, 1 \leq b \leq n, a \neq b \end{aligned} \quad (5.3)$$

4. **pointExchange**( $\Sigma$ ):

$$\begin{aligned} \Sigma' &= \langle \sigma_1, \dots, \sigma_{a-1}, \sigma_b, \sigma_{a+1}, \dots, \sigma_{b-1}, \sigma_a, \sigma_{b+1}, \dots, \sigma_n \rangle \\ &\quad \text{for } 1 \leq a \leq n, 1 \leq b \leq n, a \neq b \end{aligned} \quad (5.4)$$

In contrast to [36], these two operators are an addition to the `pathMove` and `pathExchange`. In the `pointMove` operator, the solution is changed by placing the location index  $\sigma_a$  after  $\sigma_b$  for some random  $a$  and  $b$ . On the other hand, the `pointExchange` swaps the indices  $\sigma_a$  and  $\sigma_b$ .

The particular local search technique of the employed RVNS is summarized in Algorithm 2. It tries different randomized operators in the loop for  $n^2$  times, where  $n$  is the instance size. If the operator improves the solution as the sequence  $\Sigma'$ , it is considered as the current solution  $\Sigma$ .

---

**Algorithm 2:** `localSearch( $V, \Sigma$ )`


---

**Input:**  $V = \{v_1, \dots, v_n\}$  –  $n$  locations to be visited,  $\Sigma = \langle \sigma_1, \dots, \sigma_n \rangle$  – sequence of visits to the set of  $V$ .

**Output:**  $\Sigma$  – The sequence of visits to the set of  $V$ .

---

```

1 for  $j: 1 \dots |V|^2$  do
2   switch  $j \bmod 4$  do
3     case 0 do
4       |  $\Sigma' \leftarrow \text{pointMove}(\Sigma)$ 
5     case 1 do
6       |  $\Sigma' \leftarrow \text{pointExchange}(\Sigma)$ 
7     case 2 do
8       |  $\Sigma' \leftarrow \text{pathMove}(\Sigma)$ 
9     otherwise do
10      |  $\Sigma' \leftarrow \text{pathExchange}(\Sigma)$ 
11    end
12  end
13  if  $\mathcal{L}(\Sigma') < \mathcal{L}(\Sigma)$  then
14    |  $\Sigma \leftarrow \Sigma'$ 
15  end
16 end
17 return  $\Sigma$ 

```

---



# Results

The proposed hybrid approach has been empirically evaluated on two types of benchmark instances. The first type is random instances adopted from [18] with locations fitted into a unit squared. The locations coordinates are placed randomly using a normal distribution. The size of the instances ranges from 20 to 500 locations, and the instances are grouped according to the number of locations into scenarios: Random 20, Random 50, Random 100, Random 200, Random 300, and Random 500. For each scenario, 5 random instances have been generated using seed 777. Instances in the second type of scenarios are selected from the TSP benchmark called the TSPLIB [32], which includes real-world instances with various spatial distribution and distance metrics; however, only instances with the Euclidean distance are used because of the utilized trained models. The TSPLIB scenario consists of 42 instances that are listed in Tables A.3, A.4, A.7 and A.8. An example of the instances is depicted in Fig. 6.1.

The evaluated scalability of the models also suggested limiting the size of the examined instance to 500 locations. The examined GNNs approaches [20] allocates excessively a large amount of memory for large instances. Nevertheless, the models are trained on instances from 20 to 200 locations, and the examination of scenarios with up to 500 locations provides insights into the scalability of the models.

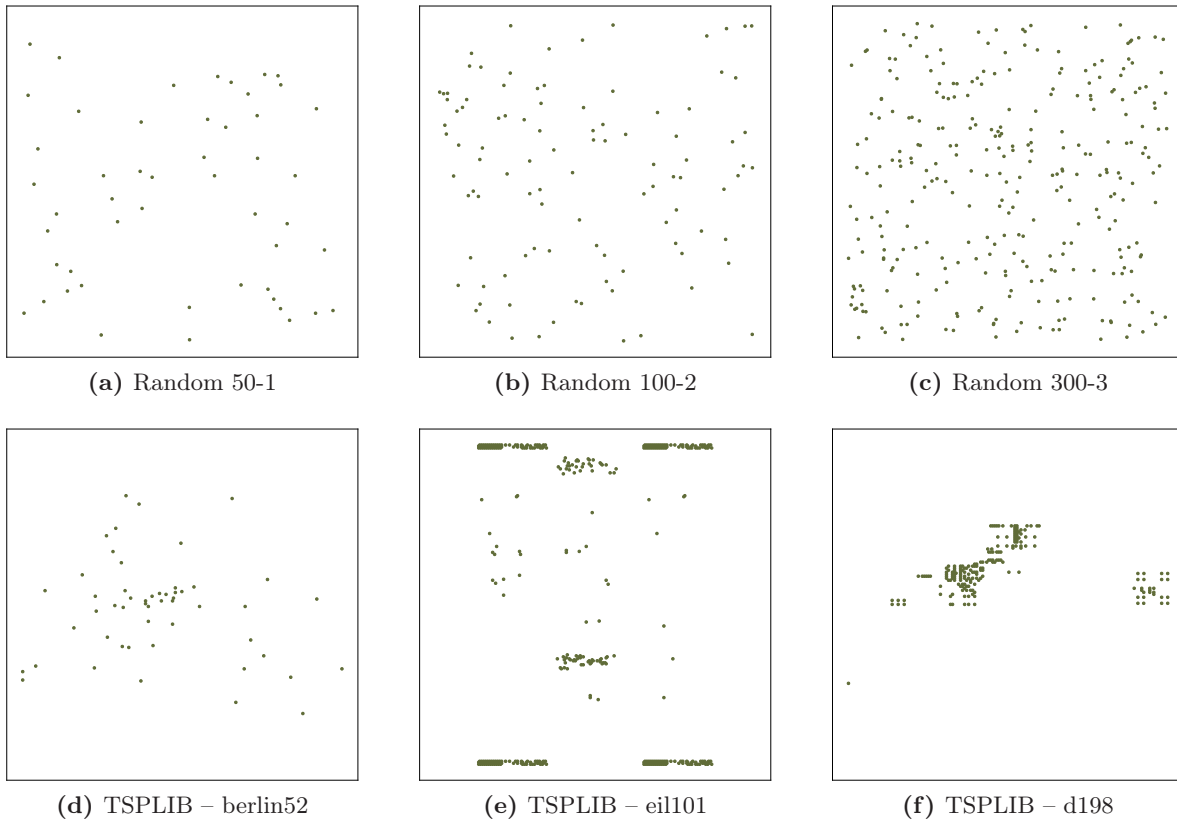
The solutions found by the proposed hybrid approach are compared with the optimal solutions determined by the Concorde solver [3]. The available implementation of the Attention Model (AM) [18] and Graph Neural Network (GNN) [20] have been used. The implementations are in Python using the PyTorch library. We further employed GSOA [16] with the available implementation in C++. It is a stochastic algorithm; therefore, each instance has been solved for 10 trials, and the mean solution cost is reported. Besides, we examined the greedy heuristic implemented in Python. All solvers have been run within the same computational environment with the Intel Core i7-9700 running at 3.0 GHz.

The solution quality of a found sequence  $\Sigma$  is reported as the relative optimality Gap determined using the optimal solution cost  $\mathcal{L}^*$ :

$$\text{Gap}(\Sigma) = \left( \frac{\mathcal{L}(\Sigma)}{\mathcal{L}^*} - 1 \right) \cdot 100 [\%], \quad (6.1)$$

where  $\mathcal{L}(\Sigma)$  is the length of the found solution or mean solution cost in the case of GSOA.





**Figure 6.1:** Example of benchmark instances scaled to a unit square, three randomly generated instances and three instances from the TSPLIB [32], `berlin52`, `eil101`, and `d198`. The black frame represents the unit square; it can be seen that `eil101` and `d198` do not resemble the randomly generated instances. Locations are clustered in `eil101` while a grid of locations with a smaller height than width can be observed for `d198`.

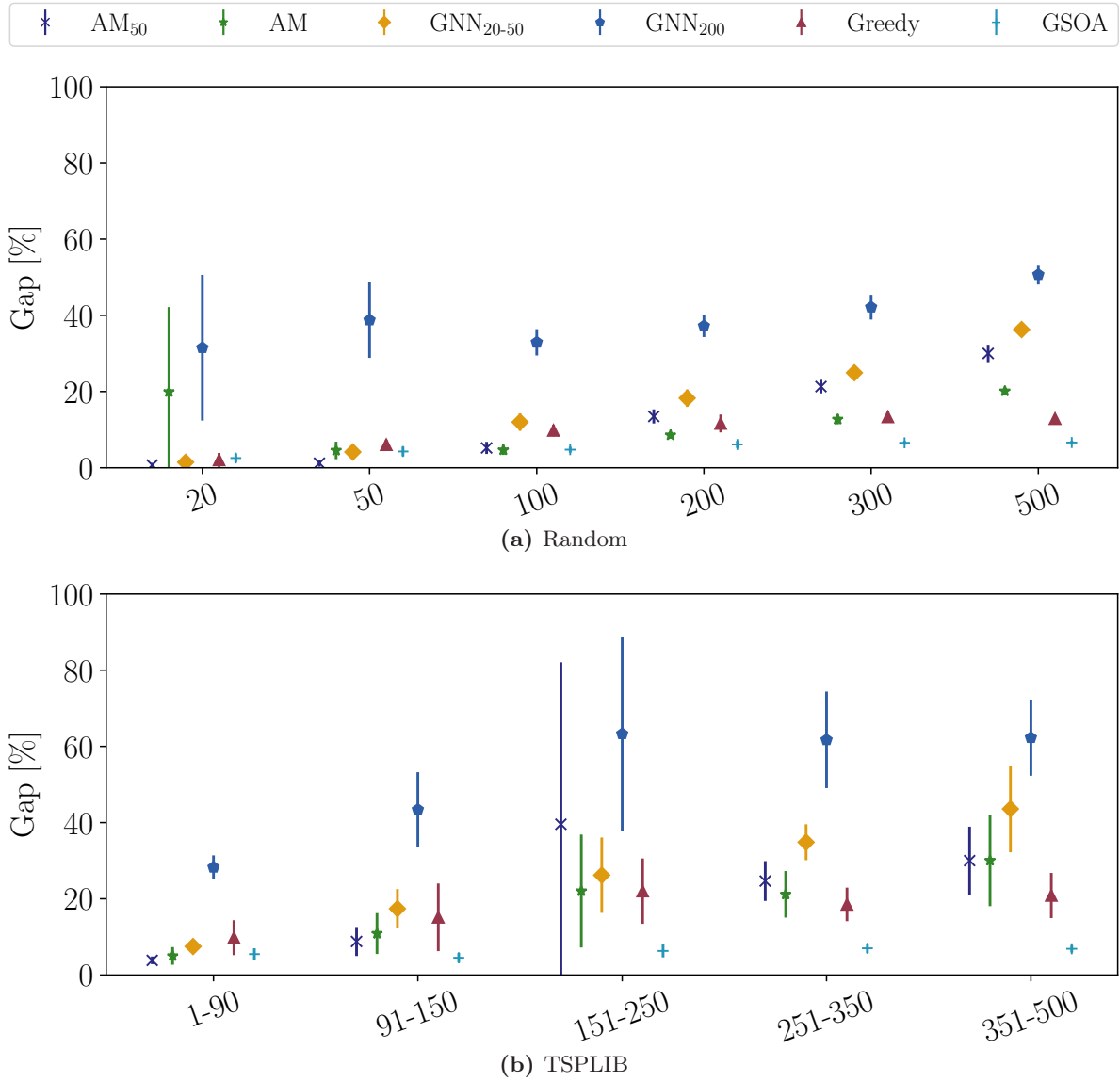
Due to the relatively high number of test instances, the instances in the Random and TSPLIB scenarios are grouped according to the number of locations, and aggregated results are reported. The Random instances are aggregated into group with 20, 50, 100, 200, 300, and 500 locations. The TSPLIB instances into groups with 1–90, 91–150, 151–250, 251–350, 351–500 locations. The aggregated results are reported as average performance indicators denoted Avg and standard deviation denoted Std.

The reported empirical results are organized as follows. Evaluation of the NN-based approaches is reported in the following section. The influence of the Set of Solutions (SoS) is examined in Section 6.2. The Feasible Solution Creators (FSCs) are examined in Section 6.3. The influence of the initial solution on the performance of the VNS-based improving heuristic of the proposed hybrid approaches is studied in Section 6.4.

## 6.1 Performance of the Existing NN-based Solvers

We consider two models for each NN-based approach. The Attention Model (AM) [18] trained on 50 locations is denoted as  $AM_{50}$ , and the AM model trained on 100 locations is denoted AM. The Graph Neural Network (GNN) model [20] trained on instances with 20–50 and 200 is

denoted  $\text{GNN}_{20-50}$  and  $\text{GNN}_{200}$ , respectively. Aggregated results on the Random and TSPLIB instances of the AM and GNN are depicted in Fig. 6.2. The full results are listed in Tables A.1 to A.4.



**Figure 6.2:** Aggregated results of the Attention Model (AM) [18] and Graph Neural Network (GNN) [20] solvers. The AM model trained on 50 and 100 locations is denoted  $\text{AM}_{50}$  and AM, respectively. The  $\text{GNN}_{20-50}$  and  $\text{GNN}_{200}$  stand for the GNN trained on 20–50 and 200 locations, respectively. The average gap is shown with the bars denoting the standard deviation.

Overall, the GSOA approach provides the best results among the examined heuristic solvers. It generalizes to instances with any spatial distribution and sizes. The  $\text{AM}_{50}$  and the  $\text{GNN}_{20-50}$  approaches provide better results than the Greedy solver, but mostly for relatively small instances. The generalization of the AM and GNN approaches is poor for increasing the size of the random instances. Because the AM model generalizes better for larger instances than the  $\text{AM}_{50}$  and both GNN, it is chosen to be a part of the proposed hybrid model pipeline in the following examinations.

## 6.2 Influence of using SoS Instances

The feasibility of the proposed Set of Solutions (SoS) presented in Section 5.1 is evaluated on the TSPLIB instances. The SoS is filled with the instances, AM embeddings, and optimal solutions found by Concorde. Since the AM is deterministic, the embeddings should match exactly its embedding in the SoS; thus, a similar solution should be the optimal path. We consider two SoS instances to validate that. The first is SoS<sub>Random</sub>, which is filled by solutions of the random instances. The second is SoS<sub>TSPLIB</sub> filled with the TSPLIB instances. For both cases, we employed greedy FSC (Section 5.3.1).

**Table 6.1:** Aggregated results for SoS<sub>Random</sub> and SoS<sub>TSPLIB</sub>

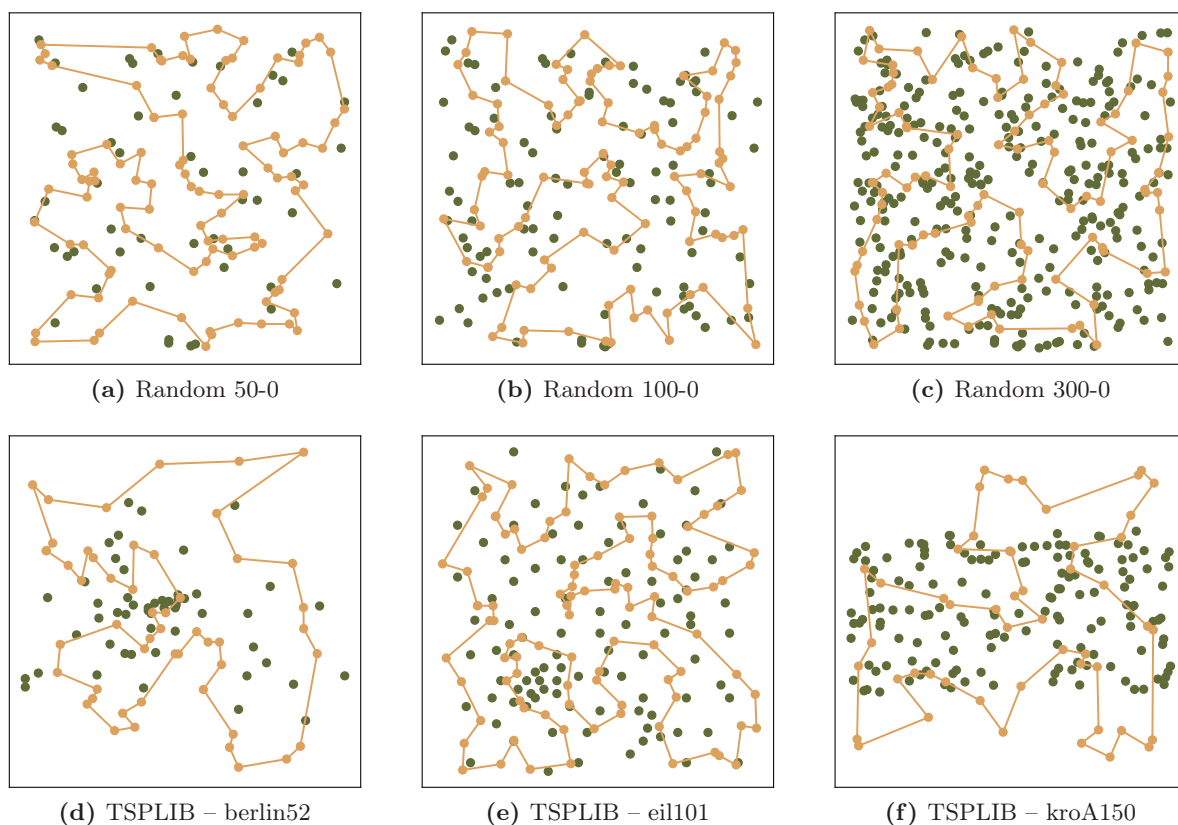
	V	SoS <sub>Random</sub>		SoS <sub>TSPLIB</sub>	
		Gap [%]		Gap [%]	
		Avg	Std	Avg	Std
TSPLIB	<b>1–90</b>	18.4	4.0	<b>0.0</b>	<b>0.0</b>
	<b>91–150</b>	17.3	4.5	<b>0.0</b>	<b>0.0</b>
	<b>151–250</b>	14.3	2.8	<b>0.0</b>	<b>0.0</b>
	<b>251–350</b>	16.6	3.2	<b>0.0</b>	<b>0.0</b>
	<b>351–500</b>	19.8	4.1	<b>0.0</b>	<b>0.0</b>

Aggregated results depicted in Table 6.1 support the expectation that the embedding can be utilized to retrieve the solution of the already stored instance as the gap is zero for querying solutions of the TSPLIB instances from the SoS<sub>TSPLIB</sub>. On the other hand, the gap for SoS<sub>Random</sub> is about 14–20%. Examples of selected input instances and the retrieved solution of the similar instance from the SoS<sub>Random</sub> are depicted in Fig. 6.3. For instances in Figs. 6.3a and 6.3d, the matched similar instances have approximately the same number of locations, and the majority of locations are concentrated in resembling positions.

## 6.3 Influence of Feasible Solution Creators

The two proposed FSCs are the Greedy approach **FSC<sub>Greedy</sub>** and the one utilizing **GSOA** labeled as **FSC<sub>GSOA</sub>**. In the evaluation of both approaches, the SoS is filled with the solutions of the random instances with 50 and 100 that are generated with the seed 42. Thus, the instances are different from the instances of the Random scenario. The aggregated results are depicted in Fig. 6.4 and detailed results can be found in Tables A.5 to A.8.

Since the AM is used to retrieve similar instances from the SoS, the results suggest the **FSC<sub>Greedy</sub>** worsens the results of the AM and Greedy heuristic used solely. A noticeable improvement can be seen for large TSPLIB instances, which might be due to the less restricting partial solution, and more locations being inserted greedily by the **FCS<sub>Greedy</sub>**. Nevertheless, its overall performance is poor. The **FCS<sub>GSOA</sub>** noticeably helps, and the initialized GSOA provides better solutions despite the lowered learning gain. Moreover, the **FCS<sub>GSOA</sub>** also leverages the ability of **GSOA** to generalize, finding better solutions even on larger instances.

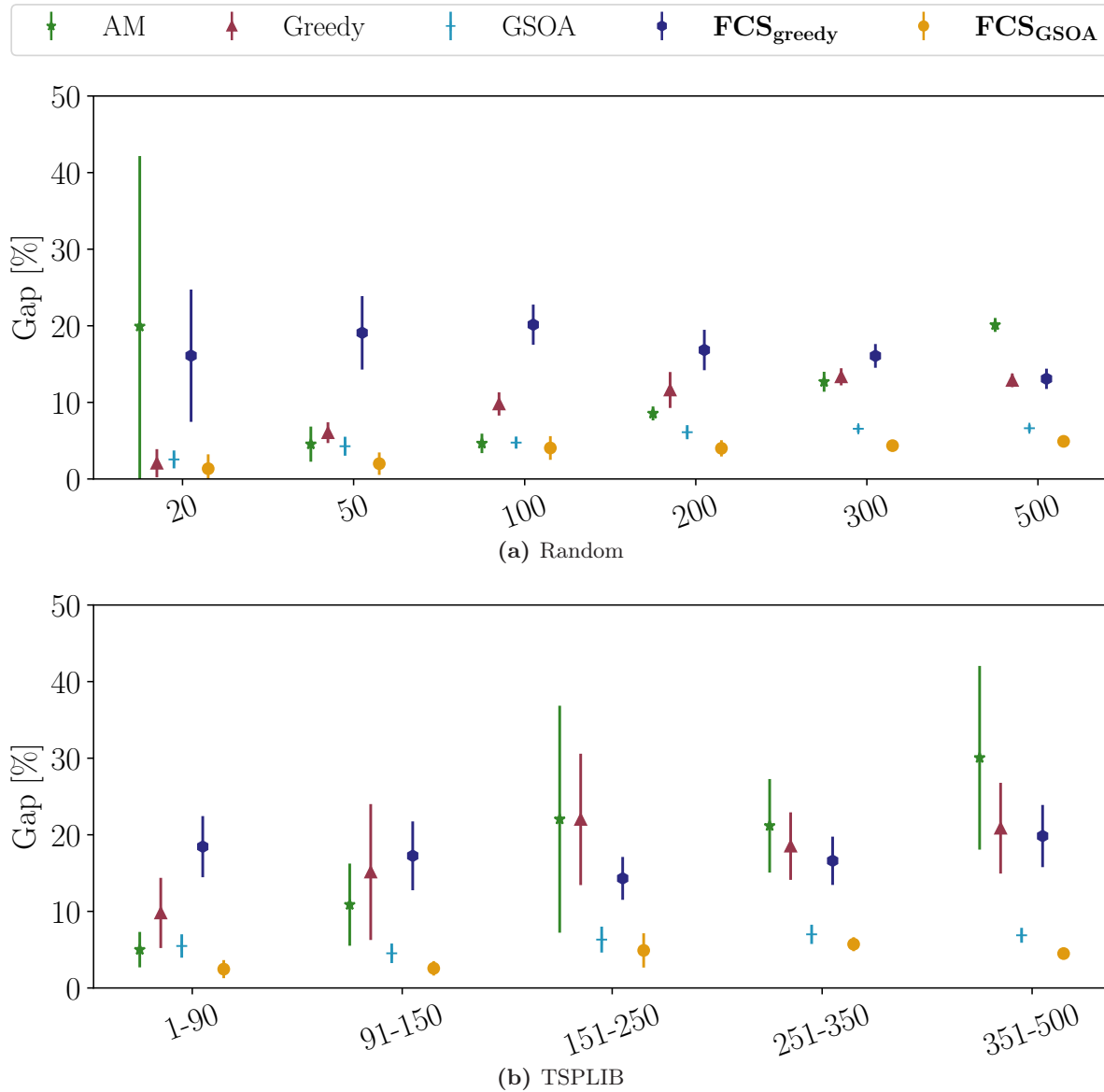


**Figure 6.3:** Examples of the retrieved similar instances from  $\text{SoS}_{\text{Random}}$  for the input instances. The green disks represent the input locations, and the orange curve is for the solution of the retrieved most similar instance.

## 6.4 Influence of the Initialization to the VNS-based Improvement Heuristic

The last performed empirical evaluation focuses on the initialization's influence of the VNS-based improving heuristic. Here, we aim to answer whether a different initialization would help faster convergence to better results of the employed VNS-based solver. Therefore, the VNS has been initialized by solutions provided by the AM learned on 100 locations, Greedy solver, one (randomly picked) solution of the GSOA, and by the proposed hybrid approaches  $\text{FSC}_{\text{greedy}}$  and  $\text{FSC}_{\text{GSOA}}$ , both with the SoS. The VNS has been limited to 2000 iterations with additional termination conditions for the maximum number of consecutive iterations without an improvement set to 100. Because of high computational requirements, the performance study of the VNS-based solver is reported only for four selected instances: Random 100-0, Random-300-0, and eil101 and kroA150 from TSPLIB, which represent instances with the number of locations closer to the learned AM and above it. Besides, the AM provides relatively good solutions for the TSPLIB instance eil101, while kroA150 is difficult for the learned AM.

The results suggest that for Random instances, the initialization is crucial, and the VNS does not converge to a similar solution. On average, the VNS is prematurely terminated before reaching the used limit of 2000 iterations. The improvement of the proposed hybrid approach with the  $\text{FSC}_{\text{GSOA}}$  is very slow, which might indicate the solution is stuck in a local optimum

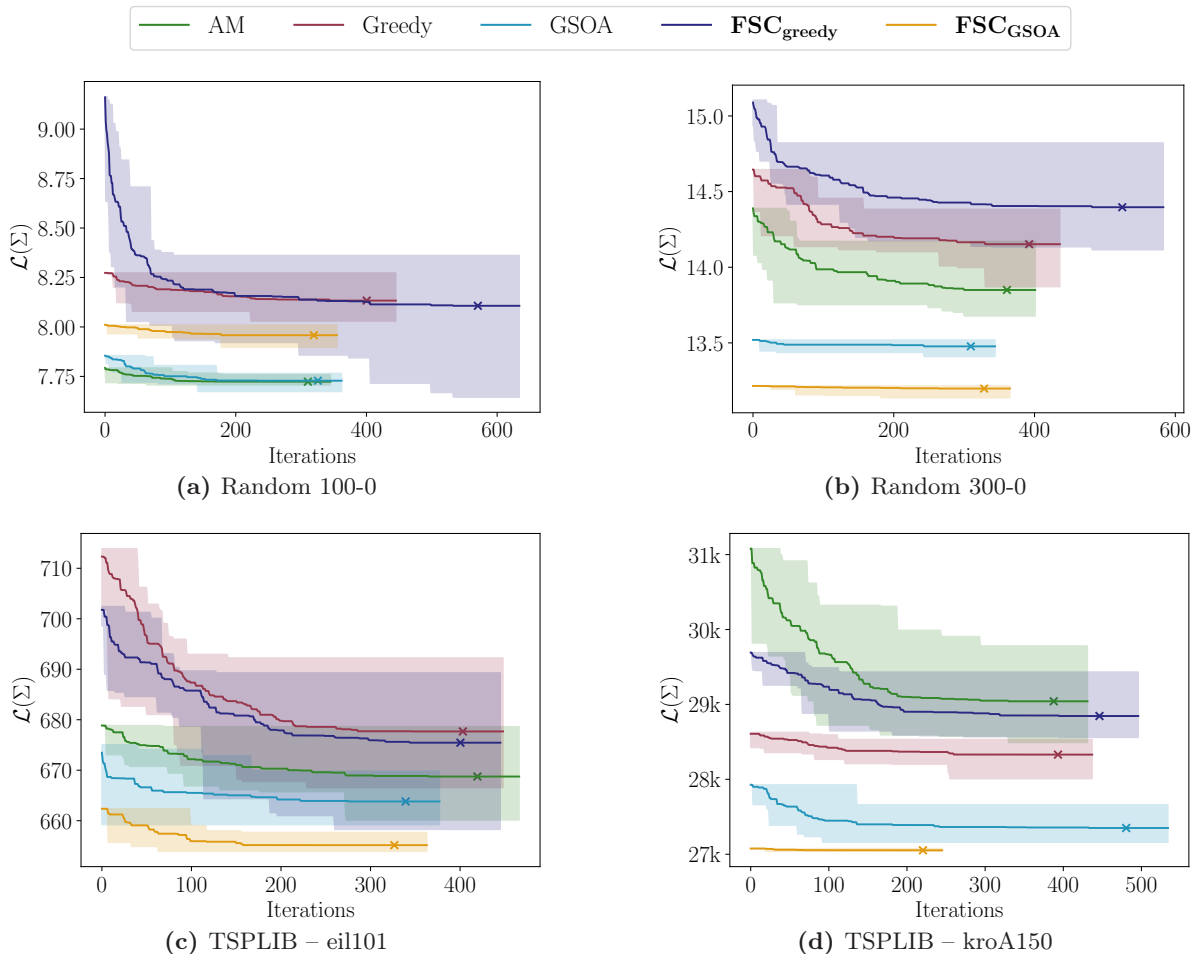


**Figure 6.4:** Aggregated results using  $\text{FCS}_{\text{greedy}}$  and  $\text{FCS}_{\text{GSOA}}$  of the Random and TSPLIB instances, and SoS filled with different random instances with 50 and 100 instances. The results of the AM, Greedy, and GSOA do not use SoS. The reported values are the average gaps per group with standard deviation visualized as the error bar.

as the average gap is about 2% for the instance kroA150, see detail result in Table A.7. However, the VNS generally improves the solutions and provides stable results.

## 6.5 Discussion

The presented results assess the existing NN-based methods and the proposed hybrid approach. The NN-based approaches (AM and GNN) provide competitive solutions only for instances the size close to the instances used for learning. The approaches do not scale with increasing instance size. Besides, we consider the approaches fail on instances from the TSPLIB. Fur-



**Figure 6.5:** Influence of different initializations to the VNS for up to 2000 iterations and termination after 100 consecutive iterations without improvement. Different Initializations of the VNS algorithm on two Random instances and two instances from TSPLIB where the TSPLIB instance eli101 is selected because the AM provides good results, whereas kroA150 is difficult. The solid bold curves represent average results over 10 runs, and the same colored area around the curve represents a difference between the best and worst solutions found at each particular iteration of the VNS. The average iteration when the VNS has been prematurely terminated because the solution has not improved is denoted by crosses.

thermore, the computational requirements are significantly higher than for Concorde providing optimal solutions. However, it might be caused by the Python implementation. On the other hand, for small random instances, and in a few cases also for small instances from the TSPLIB, the AM yields better results than the average solutions provided by GSOA.

Regarding the proposed hybrid approach, the results support the feasibility of the Set of Solutions (SoS) and using embedding of the AM to retrieve similar instances. Furthermore, the initialization using the optimal solution of the retrieved similar instance yields an improved solution found by GSOA. Although, in a few cases, the average solution cost is worse for the hybrid approach with the initialization than without it, the solutions are noticeably improved, specifically for selected large instances from the TSPLIB.

Based on the results and observations, we can answer the research question from Section 3.2.1 as follows.

- *Would usage of the NN-based instance characterization yield a similarity measure to retrieve a similar instance with a solution that would yield an improved initial solution than a traditional construction heuristic?*

The proposed Set of Solutions (SoS) based on embeddings of the Attention Model (AM) [18] allows to retrieve similar instance, which in the case of the instance already in the SoS retrieves the same instance. A solution of the similar instance used for initializing the GSOA-based solver yields noticeably better solutions than solutions found by GSOA without the initialization. Therefore, we consider the results supportive and the proposed hybrid solver vital.

# Conclusion

In the thesis, we study the well-known **TSP** by relatively recent **NN**-based approaches. Based on the survey of the existing methods, we selected **AM** and **GNN** based methods with promising results reported in the literature. However, examining their performance on instances larger than those used for learning indicates relatively poor scalability. The results support the observation that models only learned on generated random instances are not able to generalize to real-world instances of the **TSPLIB**. However, the **AM** approach is suitable for providing data abstraction, and the methods combining the **NNs** with classical heuristics showed to be promising.

The proposed hybrid approach is based on finding similar solutions using the **AM** decoder as a feature extractor providing an instance descriptor. It utilizes a **SoS** with solved instances from which the most similar solution to the query instance is determined using the L2 Norm of the instances' descriptors. A similar solution is provided to the **FSCs** based on the greedy algorithm and **GSOA**, where **GSOA** shows to provide better results. The improvement part of the proposed hybrid approach shows only a slightly fine-tuned initial solution using the combinatorial metaheuristic **VNS**. Overall, the hybrid approaches showed that the **FSC** is an important part of the pipeline, and even without an improving heuristic, the hybrid approach can improve the current methods.

Future work can exploit the modular architecture of the proposed hybrid pipeline, where alternative methods can eventually improve every part of it. For example, the most straightforward feature extraction for the Euclidean **TSP** can be based on K-means clustering. Moreover, the presented feature extractor can be trained on a new dataset generated by various distributions to address real-world instances. Moreover, rather than retrieving only a single similar instance,  $k$ -nearest neighbors can be utilized to obtain  $k$  initial solutions that can be used for the initialization of the population-based combinatorial metaheuristics.





# Bibliography

- [1] David L Applegate, Robert E Bixby, Vašek Chvátal, and William J Cook. *The Traveling Salesman Problem*. Princeton university press, 2011.
- [2] Gregory Gutin and Abraham P Punnen. *The Traveling Salesman Problem and Its Variations*, volume 12. Springer Science & Business Media, 2006.
- [3] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. CONCORDE TSP Solver, 2003. [cited May22, 2023].
- [4] Nicos 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.
- [5] Pedro Larranaga, Cindy M. H. Kuijpers, Roberto H. Murga, Inaki Inza, and Sejla Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13:129–170, 1999.
- [6] Alex Van Breedam. Improvement heuristics for the vehicle routing problem based on simulated annealing. *European Journal of Operational Research (EJOR)*, 86(3):480–490, 1995.
- [7] Pierre Hansen and Nenad Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research (EJOR)*, 130(3):449–467, 2001.
- [8] Thomas A. Feo and Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.
- [9] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.
- [10] Keld Helsgaun. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European journal of operational research*, 126(1):106–130, 2000.
- [11] K. Helsgaun. LKH, version 2.0.10, 2022. [cited May, 22 2023].

## Bibliography

- [12] John J Hopfield and David W Tank. “Neural” computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.
- [13] Sreeram VB Aiyer, Mahesan Niranjan, and Frank Fallside. A theoretical investigation into the performance of the hopfield model. *IEEE Transactions on Neural Networks*, 1(2):204–215, 1990.
- [14] Bernard Angeniol, Gael De La Croix Vaubois, and Jean-Yves Le Texier. Self-organizing feature maps and the travelling salesman problem. *Neural Networks*, 1(4):289–293, 1988.
- [15] Bernd Fritzke and Peter Wilke. FLEXMAP – a neural network for the traveling salesman problem with linear time and space complexity. In *International Joint Conference on Neural Networks (IJCNN)*, Singapore, pages 929–934, 1991.
- [16] Jan Faigl. GSOA: Growing self-organizing array - unsupervised learning for the close-enough traveling salesman problem and other routing problems. *Neurocomputing*, 312:120–134, 2018.
- [17] Zhengxuan Ling, Xinyu Tao, Yu Zhang, and Xi Chen. Solving optimization problems through fully convolutional networks: An application to the traveling salesman problem. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(12):7475–7485, 2020.
- [18] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2019.
- [19] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Step-wise deep learning models for solving routing problems. *IEEE Transactions on Industrial Informatics*, 17(7):4861–4871, 2020.
- [20] Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning TSP requires rethinking generalization. In *International Conference on Principles and Practice of Constraint Programming (CP)*, 2021.
- [21] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *European Conference on Artificial Intelligence (ECAI)*, pages 443–450. IOS Press, 2020.
- [22] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. NeuroLKH: Combining deep learning model with Lin-Kernighan-Helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems*, 34:7472–7483, 2021.
- [23] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. SIAM, 2002.
- [24] T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.
- [25] J. C. Fort. Solving a combinatorial problem via self-organizing process: An application of the Kohonen algorithm to the traveling salesman problem. *Biological Cybernetics*, 59(1):33–40, 1988.
- [26] R. Durbin and D.J. Willshaw. An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326(16):689–691, April 1987.

- [27] Samerkae Somhom, Abdolhamid Modares, and Takao Enkawa. A self-organising model for the travelling salesman problem. *Journal of the Operational Research Society*, pages 919–928, 1997.
- [28] Jan Faigl. On the performance of self-organizing maps for the non-euclidean traveling salesman problem in the polygonal domain. *Information Sciences*, 181(19):4214–4229, 2011.
- [29] Jan Faigl. Unsupervised learning-based solution of the close enough dubins orienteering problem. *Neural Computing and Applications*, 24(32):18193–18211, 2020.
- [30] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM Computing Surveys (CSUR)*, 54(10s):1–41, 2022.
- [31] Keld Helsgaun. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12, 2017.
- [32] Gerhard Reinelt. TSPLIB – a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.
- [33] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takáč. Reinforcement learning for solving the vehicle routing problem. *Advances in Neural Information Processing Systems*, 31, 2018.
- [34] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.
- [35] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research (EJOR)*, 234(3):658–673, 2014.
- [36] Robert Pěnička, Jan Faigl, Petr Váňa, and Martin Saska. Dubins orienteering problem. *IEEE Robotics and Automation Letters*, 2(2):1210–1217, 2017.

## *Bibliography*



# Detailed Results

Table A.1: Performance of the existing NN-based approaches on the Random instances – Part 1

Concorde [1]		AM <sub>50</sub> [18]		AM [18]		GNN <sub>20-50</sub> [20]		GNN <sub>200</sub> [20]		Greedy		GSOA [16]		
$\mathcal{L}^*$	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	
<b>20-0</b>	4.10	60	<b>0.0</b>	37	1.5	2 526	2.1	3 644	11.1	450	2.2	2	2.4	1
<b>20-1</b>	3.91	60	<b>0.0</b>	37	3.2	2 553	3.3	3 622	5.9	2 132	3.2	2	6.3	1
<b>20-2</b>	4.26	50	0.7	39	11.8	2 519	2.6	3 662	35.0	2 537	<b>0.0</b>	2	2.3	1
<b>20-3</b>	3.63	40	1.7	35	2.8	2 538	<b>0.0</b>	3 655	9.7	2 824	5.7	2	4.6	1
<b>20-4</b>	4.10	40	1.1	33	17.5	2 541	0.8	3 594	78.8	2 803	<b>0.0</b>	2	1.7	1
<b>50-0</b>	5.87	340	<b>2.8</b>	1 861	3.0	5 198	7.3	6 392	18.7	5 605	3.6	4	3.0	4
<b>50-1</b>	5.42	330	0.0	531	<b>0.0</b>	5 204	1.5	6 389	47.2	5 686	9.6	4	4.1	4
<b>50-2</b>	5.25	360	<b>1.9</b>	532	7.6	5 200	2.6	6 387	27.9	5 631	7.2	4	2.9	4
<b>50-3</b>	5.23	340	2.5	97	5.7	5 214	3.4	6 387	39.7	5 616	6.8	4	<b>1.7</b>	4
<b>50-4</b>	5.52	330	<b>0.0</b>	2 745	4.8	5 218	6.7	6 337	36.6	5 605	6.5	4	4.1	4
<b>100-0</b>	7.59	930	4.2	1 802	<b>2.7</b>	9 170	6.6	10 851	29.0	9 200	9.1	8	4.7	11
<b>100-1</b>	7.64	950	6.1	217	<b>5.2</b>	9 614	9.8	11 210	35.6	7 976	15.2	8	6.6	11
<b>100-2</b>	7.86	960	<b>2.3</b>	177	6.6	9 724	12.8	10 741	28.9	6 113	7.8	8	4.6	12
<b>100-3</b>	7.77	960	5.4	2 993	<b>4.3</b>	9 760	13.8	10 709	37.9	1 565	8.7	8	5.1	12
<b>100-4</b>	7.68	950	6.9	1 996	5.2	9 736	11.6	10 787	37.5	1 352	9.6	8	<b>3.5</b>	11

Table A.2: Performance of the existing NN-based approaches on the Random instances – Part 2

Concorde [1]		AM <sub>50</sub> [18]		AM [18]		GNN <sub>20-50</sub> [20]		GNN <sub>200</sub> [20]		Greedy		GSOA [16]		
$\mathcal{L}^*$	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	
<b>200-0</b>	10.52	2 010	16.3	1 649	9.0	23 113	20.3	24 311	38.7	23 416	<b>8.5</b>	17	8.8	43
<b>200-1</b>	10.79	2 210	17.1	7 240	9.7	23 113	20.3	24 320	44.4	23 512	13.5	25	<b>6.9</b>	43
<b>200-2</b>	10.87	2 120	13.7	15 165	8.5	23 183	16.1	24 326	41.1	14 265	8.2	16	<b>5.7</b>	43
<b>200-3</b>	10.77	2 100	12.0	19 311	<b>6.3</b>	23 189	13.9	24 400	32.2	6 286	10.5	17	6.5	43
<b>200-4</b>	10.42	2 630	13.7	4 583	8.1	23 218	21.0	24 574	39.0	13 841	12.0	17	<b>5.5</b>	42
<b>300-0</b>	12.85	3 150	23.7	4 680	11.9	41 571	24.7	43 451	47.2	41 927	13.9	26	<b>6.8</b>	102
<b>300-1</b>	12.82	3 120	20.8	24 588	12.0	41 561	27.3	43 374	42.2	42 524	13.7	27	<b>7.9</b>	102
<b>300-2</b>	12.96	3 310	18.0	8 238	10.6	41 512	22.3	43 602	37.7	42 141	10.2	26	<b>7.0</b>	102
<b>300-3</b>	12.87	3 300	21.0	5 456	13.4	41 478	27.1	43 269	40.4	42 078	15.3	26	<b>6.5</b>	102
<b>300-4</b>	13.16	3 350	23.7	9 997	14.2	41 587	21.6	43 562	45.1	42 097	14.8	27	<b>5.8</b>	113
<b>500-0</b>	16.70	6 510	32.1	29 977	18.9	68 168	35.0	70 321	51.9	58 826	14.0	48	<b>6.4</b>	284
<b>500-1</b>	16.48	6 310	30.7	21 943	20.3	68 400	40.1	70 796	49.2	68 896	11.4	46	<b>6.7</b>	303
<b>500-2</b>	16.49	6 650	23.8	18 329	18.4	68 142	32.1	70 244	52.8	64 654	13.3	47	<b>5.7</b>	287
<b>500-3</b>	16.51	6 550	31.2	26 119	20.3	68 174	34.8	70 150	54.2	68 926	12.6	49	<b>6.6</b>	291
<b>500-4</b>	16.44	8 000	26.6	50 092	20.2	68 164	34.9	70 481	56.7	68 455	13.4	49	<b>7.7</b>	288



Table A.3: Performance of the existing NN-based approaches on the TSPLIB instances – Part 1

Concorde [1]		AM <sub>50</sub> [18]		AM [18]		GNN <sub>20-50</sub> [20]		GNN <sub>200</sub> [20]		Greedy		GSOA [16]	
$\mathcal{L}^*$	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]
eil51	429	30	<b>2.4</b>	5 313	2.7	5 368	6.6	6 410	25.6	2 116	8.5	4	5.4
berlin52	7 544	40	6.3	5 393	10.2	5 376	10.6	6 472	32.7	3 287	<b>4.5</b>	4	9.3
st70	678	60	3.6	7 259	<b>2.3</b>	7 233	8.5	6 326	23.1	3 913	4.9	6	3.2
pr76	108 159	430	<b>3.0</b>	7 806	5.5	7 772	5.0	5 207	31.6	7 589	19.3	6	4.2
eil76	545	30	<b>3.9</b>	7 780	4.2	7 866	6.7	8 601	28.4	1 549	11.8	6	5.3
rat99	1 219	150	14.9	9 890	11.5	9 808	22.9	9 723	108.9	5 736	31.4	8	<b>5.9</b>
kroD100	21 294	160	9.2	9 954	6.4	9 886	16.6	10 006	40.7	9 774	9.9	8	<b>4.3</b>
kroC100	20 751	80	5.5	9 962	6.6	9 895	10.9	10 230	42.6	9 696	4.9	8	<b>1.7</b>
kroB100	22 139	180	<b>2.7</b>	9 994	9.6	9 934	15.1	10 089	39.2	1 719	11.2	8	5.3
kroA100	21 285	110	4.0	9 958	10.2	9 929	22.0	10 092	40.3	3 243	5.2	8	<b>3.1</b>
kroE100	22 069	220	5.7	9 953	8.6	9 893	19.9	3 247	60.7	10 057	7.8	8	<b>4.7</b>
rd100	7 910	70	6.7	9 981	7.6	9 918	12.7	9 019	24.5	5 445	12.3	8	<b>5.8</b>
eil101	642	70	6.1	10 021	<b>5.7</b>	9 983	9.3	10 881	27.7	4 732	11.2	8	5.8
lin105	14 383	60	15.7	10 369	7.0	10 334	24.8	10 541	45.0	10 260	25.9	9	<b>2.9</b>
pr107	44 302	120	12.4	10 604	52.3	10 566	11.9	4 503	37.7	2 383	16.9	9	<b>1.4</b>
pr124	59 031	290	6.8	12 106	5.5	12 054	17.9	10 788	45.7	780	13.8	10	<b>3.3</b>
bier127	118 294	180	15.3	12 246	8.6	12 238	30.2	13 157	45.0	11 554	12.0	10	<b>6.1</b>
ch130	6 111	200	7.0	15 465	<b>5.2</b>	15 421	12.8	16 364	34.0	15 121	6.3	11	5.4
pr136	96 771	400	<b>5.0</b>	16 307	9.2	16 206	12.5	15 811	43.3	2 993	36.3	11	6.2
pr144	58 535	260	<b>4.2</b>	17 176	8.2	17 218	8.5	16 327	31.4	11 668	45.1	12	5.3
kroA150	26 525	410	16.2	17 828	17.2	17 735	24.2	17 256	39.6	5 104	7.9	12	<b>5.8</b>
ch150	6 531	330	11.0	17 771	6.0	17 663	19.0	18 574	32.0	17 336	9.4	13	<b>4.9</b>

Table A.4: Performance of the existing NN-based approaches on the TSPLIB instances – Part 2

	Concorde [1]		AM <sub>50</sub> [18]		AM [18]		GNN <sub>20-50</sub> [20]		GNN <sub>200</sub> [20]		Greedy		GSOA [16]	
	$\mathcal{L}^*$	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]
<b>kroB150</b>	26 127	490	10.0	17 871	10.5	17 772	21.9	17 585	43.2	10 416	4.9	12	<b>3.6</b>	25
<b>pr152</b>	73 684	790	7.8	18 059	10.8	17 999	16.4	9 664	33.4	4 645	25.0	13	<b>4.1</b>	26
<b>u159</b>	42 076	130	12.7	18 881	15.4	18 766	24.8	14 201	50.8	5 742	25.6	13	<b>5.5</b>	27
<b>rat195</b>	2 334	3 020	22.9	22 815	15.5	22 872	36.2	14 328	175.4	13 442	31.9	17	<b>8.7</b>	43
<b>d198</b>	15 809	1 090	230.9	23 126	88.7	22 993	59.2	23 982	66.1	22 000	12.3	18	<b>8.2</b>	42
<b>kroB200</b>	29 440	290	17.2	23 480	15.7	23 288	21.3	17 955	50.2	7 271	11.8	17	<b>5.1</b>	43
<b>kroA200</b>	29 369	560	19.4	23 506	15.7	23 281	22.8	11 984	47.8	8 693	8.4	16	<b>5.7</b>	43
<b>tsp225</b>	3 859	1 120	22.6	26 281	14.0	26 183	27.7	21 867	52.5	9 213	17.0	20	<b>7.2</b>	56
<b>ts225</b>	126 646	3 540	15.6	26 252	13.7	26 161	17.7	19 946	42.2	11 253	37.2	19	<b>8.8</b>	55
<b>pr226</b>	80 370	340	7.0	26 294	8.8	26 373	9.7	18 232	51.1	7 585	28.8	26	<b>3.6</b>	58
<b>g1262</b>	2 389	1 490	17.5	36 846	11.9	36 686	24.9	37 849	39.5	9 702	9.5	23	<b>5.9</b>	81
<b>pr264</b>	49 135	330	28.3	37 099	30.0	37 218	36.0	31 124	68.2	17 018	16.5	24	<b>6.6</b>	83
<b>a280</b>	2 588	540	27.9	38 402	22.8	38 523	40.8	39 493	52.3	23 258	19.7	26	<b>10.1</b>	93
<b>pr299</b>	48 195	2 020	30.8	41 756	25.9	41 916	39.6	16 705	76.9	17 989	26.3	27	<b>5.3</b>	106
<b>lin318</b>	42 043	930	18.7	44 379	15.2	44 473	33.0	42 232	71.7	37 023	20.6	28	<b>7.1</b>	119
<b>rd400</b>	15 278	20 460	22.8	55 459	16.1	55 188	31.3	41 225	47.4	20 579	10.9	37	<b>6.5</b>	196
<b>f1417</b>	11 915	6 020	14.9	57 577	20.4	57 613	36.7	58 698	75.0	9 461	16.0	38	<b>5.6</b>	205
<b>pr439</b>	107 215	15 700	32.0	60 585	49.6	60 542	53.4	48 459	74.6	58 343	28.0	42	<b>7.2</b>	232
<b>pcb442</b>	50 784	7 040	32.9	61 140	23.7	61 000	34.3	13 553	54.8	25 464	25.0	42	<b>9.0</b>	232
<b>d493</b>	35 022	23 530	47.4	67 648	40.5	67 980	62.2	68 630	59.7	32 961	24.4	47	<b>6.1</b>	302

**Table A.5:** Performance of the proposed hybrid approaches on the Random instances – Part 1

Concorde [1]		AM [18]		Greedy		GSOA [16]		FCS <sub>greedy</sub>		FCS <sub>GSOA</sub>		
$\mathcal{L}^*$	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	
<b>20-0</b>	4.10	60	1.5	2 526	2.2	2	2.4	1	6.9	739	<b>0.0</b>	1
<b>20-1</b>	3.91	60	3.2	2 553	3.2	2	6.3	1	7.3	708	<b>0.0</b>	2
<b>20-2</b>	4.26	50	11.8	2 519	<b>0.0</b>	2	2.3	1	11.4	731	1.7	1
<b>20-3</b>	3.63	40	<b>2.8</b>	2 538	5.7	2	4.6	1	5.7	714	5.9	1
<b>20-4</b>	4.10	40	17.5	2 541	<b>0.0</b>	2	1.7	1	19.3	730	5.8	1
<b>50-0</b>	5.87	340	3.0	5 198	3.6	4	3.0	4	16.4	743	<b>2.0</b>	4
<b>50-1</b>	5.42	330	<b>0.0</b>	5 204	9.6	4	4.1	4	19.9	733	0.0	4
<b>50-2</b>	5.25	360	7.6	5 200	7.2	4	2.9	4	9.3	733	<b>1.7</b>	3
<b>50-3</b>	5.23	340	5.7	5 214	6.8	4	1.7	4	25.2	750	<b>1.0</b>	4
<b>50-4</b>	5.52	330	4.8	5 218	6.5	4	4.1	4	17.9	733	<b>0.9</b>	4
<b>100-0</b>	7.59	930	<b>2.7</b>	9 170	9.1	8	4.7	11	20.8	931	5.6	13
<b>100-1</b>	7.64	950	<b>5.2</b>	9 614	15.2	8	6.6	11	18.6	946	6.4	14
<b>100-2</b>	7.86	960	6.6	9 724	7.8	8	4.6	12	26.0	930	<b>4.3</b>	13
<b>100-3</b>	7.77	960	<b>4.3</b>	9 760	8.7	8	5.1	12	19.8	945	6.8	13
<b>100-4</b>	7.68	950	5.2	9 736	9.6	8	3.5	11	18.6	933	<b>1.9</b>	13

**Table A.6:** Performance of the proposed hybrid approaches on the Random instances – Part 2

	Concorde [1]		AM [18]		Greedy		GSOA [16]		FCS <sub>greedy</sub>		FCS <sub>GSOA</sub>	
	$\mathcal{L}^*$	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]
<b>200-0</b>	10.52	2 010	9.0	23 113	8.5	17	8.8	43	19.8	934	<b>5.6</b>	49
<b>200-1</b>	10.79	2 210	9.7	23 113	13.5	25	6.9	43	17.7	939	<b>5.4</b>	49
<b>200-2</b>	10.87	2 120	8.5	23 183	8.2	16	5.7	43	11.9	941	<b>3.5</b>	49
<b>200-3</b>	10.77	2 100	6.3	23 189	10.5	17	6.5	43	14.9	894	<b>3.7</b>	47
<b>200-4</b>	10.42	2 630	8.1	23 218	12.0	17	5.5	42	19.0	942	<b>4.4</b>	48
<b>300-0</b>	12.85	3 150	11.9	41 571	13.9	26	6.8	102	17.5	1 366	<b>2.8</b>	114
<b>300-100-100-1</b>	12.82	3 120	12.0	41 561	13.7	27	7.9	102	18.4	1 448	<b>4.9</b>	116
<b>300-2</b>	12.96	3 310	10.6	41 512	10.2	26	7.0	102	16.3	1 453	<b>4.4</b>	113
<b>300-3</b>	12.87	3 300	13.4	41 478	15.3	26	6.5	102	17.8	1 461	<b>4.1</b>	114
<b>300-4</b>	13.16	3 350	14.2	41 587	14.8	27	5.8	113	15.4	1 491	<b>4.9</b>	114
<b>500-0</b>	16.70	6 510	18.9	68 168	14.0	48	6.4	284	14.5	1 445	<b>4.8</b>	353
<b>500-1</b>	16.48	6 310	20.3	68 400	11.4	46	6.7	303	14.7	1 490	<b>5.4</b>	326
<b>500-2</b>	16.49	6 650	18.4	68 142	13.3	47	5.7	287	15.2	1 502	<b>4.4</b>	347
<b>500-3</b>	16.51	6 550	20.3	68 174	12.6	49	6.6	291	12.8	1 488	<b>4.8</b>	323
<b>500-4</b>	16.44	8 000	20.2	68 164	13.4	49	7.7	288	11.8	1 492	<b>5.1</b>	327

Table A.7: Performance of the proposed hybrid approaches on the TSPLIB instances – Part 1

	Concorde [1]		AM [18]		Greedy		GSOA [16]		FCS <sub>greedy</sub>		FCS <sub>GSOA</sub>	
	$\mathcal{L}^*$	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]
eil51	429	30	2.7	5 368	8.5	4	5.4	4	14.2	752	<b>2.3</b>	4
berlin52	7 544	40	10.2	5 376	4.5	4	9.3	4	19.4	741	<b>2.5</b>	3
st70	678	60	2.3	7 233	4.9	6	3.2	7	27.5	953	<b>1.4</b>	7
pr76	108 159	430	5.5	7 772	19.3	6	<b>4.2</b>	8	16.9	962	5.4	8
eil76	545	30	4.2	7 866	11.8	6	5.3	8	14.3	934	<b>0.8</b>	8
rat99	1 219	150	11.5	9 808	31.4	8	5.9	12	21.8	935	<b>3.5</b>	13
kroD100	21 294	160	6.4	9 886	9.9	8	4.3	12	21.2	945	<b>2.8</b>	12
kroC100	20 751	80	6.6	9 895	4.9	8	1.7	12	12.1	879	<b>1.0</b>	12
kroB100	22 139	180	9.6	9 934	11.2	8	5.3	12	19.2	955	<b>2.0</b>	13
kroA100	21 285	110	10.2	9 929	5.2	8	3.1	12	21.1	942	<b>0.9</b>	12
kroE100	22 069	220	8.6	9 893	7.8	8	<b>4.7</b>	11	12.1	885	5.4	12
rd100	7 910	70	7.6	9 918	12.3	8	5.8	12	13.5	933	<b>3.1</b>	13
eil101	642	70	5.7	9 983	11.2	8	5.8	13	9.4	876	<b>3.2</b>	13
lin105	14 383	60	7.0	10 334	25.9	9	2.9	13	18.0	944	<b>0.8</b>	14
pr107	44 302	120	52.3	10 566	16.9	9	1.4	13	12.4	946	<b>0.8</b>	14
pr124	59 031	290	5.5	12 054	13.8	10	<b>3.3</b>	17	27.8	941	4.2	17
bier127	118 294	180	8.6	12 238	12.0	10	6.1	19	12.9	951	<b>2.5</b>	21
ch130	6 111	200	5.2	15 421	6.3	11	5.4	19	17.6	882	<b>2.2</b>	21
pr136	96 771	400	9.2	16 206	36.3	11	6.2	21	19.6	946	<b>3.7</b>	23
pr144	58 535	260	8.2	17 218	45.1	12	5.3	23	25.2	951	<b>3.2</b>	22
kroA150	26 525	410	17.2	17 735	7.9	12	5.8	24	11.9	950	<b>2.1</b>	27
ch150	6 531	330	6.0	17 663	9.4	13	4.9	25	21.4	949	<b>2.6</b>	28

Table A.8: Performance of the proposed hybrid approaches on the TSPLIB instances – Part 2

	Concorde [1]		AM [18]		Greedy		GSOA [16]		FCS <sub>greedy</sub>		FCS <sub>GSOA</sub>	
	$\mathcal{L}^*$	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]	Gap [%]	$t$ [ms]
<b>kroB150</b>	26 127	490	10.5	17 772	4.9	12	3.6	25	13.3	932	<b>2.4</b>	27
<b>pr152</b>	73 684	790	10.8	17 999	25.0	13	<b>4.1</b>	26	6.3	947	4.5	28
<b>u159</b>	42 076	130	15.4	18 766	25.6	13	5.5	27	14.2	945	<b>4.4</b>	29
<b>rat195</b>	2 334	3 020	15.5	22 872	31.9	17	8.7	43	20.8	953	<b>7.0</b>	46
<b>d198</b>	15 809	1 090	88.7	22 993	12.3	18	8.2	42	11.2	946	<b>7.2</b>	45
<b>kroB200</b>	29 440	290	15.7	23 288	11.8	17	5.1	43	14.6	951	<b>2.5</b>	48
<b>kroA200</b>	29 369	560	15.7	23 281	8.4	16	5.7	43	16.8	945	<b>1.4</b>	47
<b>tsp225</b>	3 859	1 120	14.0	26 183	17.0	20	7.2	56	14.7	892	<b>5.1</b>	60
<b>ts225</b>	126 646	3 540	13.7	26 161	37.2	19	<b>8.8</b>	55	17.3	954	10.5	59
<b>pr226</b>	80 370	340	8.8	26 373	28.8	26	3.6	58	13.0	971	<b>1.7</b>	58
<b>gil262</b>	2 389	1 490	11.9	36 686	9.5	23	5.9	81	9.5	1 474	<b>4.4</b>	85
<b>pr264</b>	49 135	330	30.0	37 218	16.5	24	6.6	83	17.8	1 474	<b>5.0</b>	86
<b>a280</b>	2 588	540	22.8	38 523	19.7	26	10.1	93	20.7	1 372	<b>7.7</b>	98
<b>pr299</b>	48 195	2 020	25.9	41 916	26.3	27	<b>5.3</b>	106	19.3	1 479	5.5	110
<b>lin318</b>	42 043	930	15.2	44 473	20.6	28	7.1	119	15.9	1 483	<b>6.0</b>	129
<b>rd400</b>	15 278	20 460	16.1	55 188	10.9	37	6.5	196	13.8	1 484	<b>3.5</b>	220
<b>fl417</b>	11 915	6 020	20.4	57 613	16.0	38	5.6	205	29.9	1 500	<b>4.0</b>	214
<b>pr439</b>	107 215	15 700	49.6	60 542	28.0	42	7.2	232	18.2	1 482	<b>6.4</b>	243
<b>pcb442</b>	50 784	7 040	23.7	61 000	25.0	42	9.0	232	19.9	1 494	<b>4.5</b>	245
<b>d493</b>	35 022	23 530	40.5	67 980	24.4	47	6.1	302	17.4	1 504	<b>4.1</b>	317

