THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Attention On What Is Important: Improving Neural Encoders for Routing Problems

Attila Lischka

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2024

Attention On What Is Important:
Improving Neural Encoders for Routing Problems

Attila Lischka

This thesis has been prepared using LaTeX.

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000
www.chalmers.se

Printed by Chalmers Reproservice
Gothenburg, Sweden, September 2024

To my family and friends.

Abstract

Virtually all deep learning-based frameworks trying to solve routing problems have a neural encoder architecture. In this work, we explore the sparsification of graphs representing instances of routing problems. By this sparsification, we allow the neural encoder architectures of learning-based frameworks to focus on the parts of the routing problem that are most promising to be part of the problem solution. As a result, the encoders can produce better encodings that represent the problems in the neural framework. Since these good problem representations are fundamental for the overall learning pipeline, good encodings improve the overall performance.

In particular, in this thesis, we focus on graph neural network (GNN) and transformer encoders applied to instances of the traveling salesman problem (TSP). We propose two different procedures to determine the most promising edges of a TSP, i.e., the edges that are likely to be part of the optimal TSP tour. The first method is the simple k-nearest neighbor heuristic, where each node in the TSP instance is only connected to the k closest other nodes after sparsification. The second method is based on minimum spanning trees (MSTs) and offers the advantage of guaranteeing connected sparse graphs.

Furthermore, we propose ensemble methods of different sparsification levels. This means that each TSP instance is represented several times, each time as a graph with either more or less edges of the original TSP graph being kept. By combining very sparse graphs with only the most promising edges and dense graphs with a high amount of edges, we allow the encoder architecture to focus on the most important parts of the problem only while minimizing the risk of completely deleting optimal TSP tour edges in the sparsification process. The encodings produced on the TSP graphs of different sparsification levels are merged afterwards, creating encodings that can be incorporated easily into existing learning-based routing frameworks.

Keywords: Machine Learning, Traveling Salesman Problem, Vehicle Routing, Graph Neural Networks, Transformers, Combinatorial Optimization, Graph Sparsification

List of Publications

This thesis is based on the following publications:

[A] **Attila Lischka**, Jiaming Wu, Rafael Basso, Morteza Haghir Chehreghani, Balázs Kulcsár, "Less Is More – On the Importance of Sparsification for Transformers and Graph Neural Networks for TSP". Preprint.

Other publications by the author, not included in this thesis, are:

[B] Fangting Zhou, **Attila Lischka**, Balázs Kulcsár, Jiaming Wu, Morteza Haghir Chehreghani, "Learning for routing: A guided review of recent developments and future directions". In pipeline..

Acknowledgments

My biggest thanks go to my supervisors Prof. Balázs Kulcsár, Prof. Morteza Haghir Chehreghani, and Dr. Jiaming Wu. Without their help and encouragement, I would not have finished this degree. I deeply appreciate all the interesting discussions we had. They significantly contributed to guiding the direction of our research.

I would also like to express my gratitude to the Swedish Electromobility Centre (SEC) which provided funding for this thesis through the research project "LEAR: Robust LEArning methods for electric vehicle Route selection". Furthermore, I want to acknowledge that computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) at Chalmers e-Commons partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

Acronyms

CVRP:	Capacitated Vehicle Routing Problem
GAT:	Graph Attention Network
GCN:	Graph Convolutional Network
GNN:	Graph Neural Network
MCTS:	Monte Carlo Tree Search
MLP:	Multilayer Perceptron
MST:	Minimum Spanning Tree
RL:	Reinforcement Learning
SL:	Supervised Learning
TSP:	Traveling Salesman Problem
VRP:	Vehicle Routing Problem

Contents

Abstract i

List of Papers iii

Acknowledgements v

Acronyms v

I Overview 1

1 Introduction 3
1.1 Motivation 3
1.2 Contributions 5
1.3 Thesis outline 7

2 Background 9
2.1 Traveling Salesman Problem 9
Capacitated Vehicle Routing Problem 10
Graph Neural Networks 11
Limitations of Graph Neural Networks 12
2.2 Transformer Networks 12

2.3 Minimum Spanning Trees 12
1-Trees 13

3 Related Work **15**
3.1 Learning to Route 15
Construction-based Approaches 16
Improvement-based Approaches 18
3.2 Sparsifying the TSP 20

4 Methodology - Sparsifying the TSP **23**
4.1 Sparsification Methods 23
4.2 Sparse Graph Ensembles 29

5 Summary of included papers **33**
5.1 Paper A 33

6 Concluding Remarks and Future Work **35**
6.1 Conclusion 35
6.2 Future Work 36

References **39**

II Papers **45**

A Less Is More **A1**
1 Introduction A3
2 Related Work A7
2.1 Learn to Route A7
2.2 Sparsification for Routing A9
3 Preliminaries A9
3.1 Graph Neural Networks A9
3.2 Travelling Salesman Problem A10
4 Methodology A10
4.1 Making the TSP Sparse: The Sparsification Process . . A10
4.2 A Sparsification Based Framework for Learning to Route A14
4.3 Other Sparsification Methods A16

5 Experiments A16
5.1 Optimal Edge Retention Capability of Sparsification Methods A16
5.2 Sparse Graphs for Encoders Evaluation A17
5.3 Results - GNNs A19
5.4 Results - Transformers A22
6 Conclusion and Future Work A24
1 Data Distributions A25
2 Experiments - Setup A25
3 Data Augmentation A29
4 Experiments - Preprocessing Times A29
5 Experiments - Ensembles A31
References A31

Part I

Overview

CHAPTER 1

Introduction

1.1 Motivation

Vehicle routing problems are examples of optimization problems with many use cases in real-world problems such as logistics or chip designing. For example, a delivery service might be interested in serving all its customers while not overloading their delivery vehicles, starting and ending their journey at the delivery center, and minimizing the traveled distance. This is an example of the capacitated vehicle routing problem (CVRP). Another example is the traveling salesman problem (TSP) in which a set of customers is given which shall be visited by an agent in an order that minimizes the traveled distance.

Despite their omnipresence, routing problems are challenging to solve as they belong to the class of NP-hard problems. Since no efficient algorithms are known for solving NP-hard problems, heuristics play an important role in approximating good solutions. In recent years, machine learning has arisen as a powerful trade-off to build algorithms that are fast while still achieving good solution quality for routing problems.

To process the information inherent to a routing problem instance, it has to be captured and *encoded* by a suitable neural architecture. Routing problems

can be interpreted as graph problems, where, e.g., customers of a delivery company are represented as graph nodes, and the roads between the customers are represented as graph edges. As a result, graph neural networks (GNNs), a class of neural architectures designed to operate on graph-structured data, are a straightforward choice to serve as an encoder architecture for routing problems. Similarly, transformer models (that have been shown to be related to GNNs) have also been used successfully to generate encodings for routing problems in the recent past.

GNNs are known to exploit structural information when applied to underlying input graph instances [1], [2]. In routing problems, however, the input graph often carries little structural information. This is because in principle it is possible to travel between any pair of nodes in the graph which means that the graph is *complete*. Therefore, graph neural networks are not able to meaningfully extract information about promising node neighborhoods in their internal message-passing operations. In fact, the GNN will perform aggregation operations over the complete graph for every single node when performing message-passing, resulting in the exact same neighborhood information and therefore similar node encodings for every node. As a result, a framework based on these encodings will struggle to differentiate between them and structure them meaningfully to generate a valid solution. We visualize this concept with a toy example in Figure 1.1. In the figure, we have two dense graphs to the left and two sparse graphs to the right. The left dense and the left sparse graphs show the initializations of the graphs that are passed to a GNN. In the initialization, each node has its own (unique) initial feature vector which is represented as a color in the figure. The right dense and the right sparse graphs show the results of performing one message-passing operation by the graph neural network. After this message-passing operation, each node feature vector contains information on all its neighboring node feature vectors as well. We note that in the case of the dense graph, each node shares the same five colors as it was flooded with information from all graph nodes. This is not the case in the sparse graph, where each node only has three colors. We note that the color proportion of the initialization stays "larger" after the message-passing compared to the color proportions representing neighboring nodes (e.g., the node at the very bottom of the second graph is 50% white, whereas each other color only covers a smaller proportion in this node). This is because, typically, the old node feature vector is combined with the infor-

mation of the aggregation in a weighted manner emphasizing the prior node information.

We now interpret the graphs in Figure 1.1 as TSP instances and show how the sparse graph leads to better encodings. We note again that in the graphs to the right of the figure, only the optimal TSP edges are part of the graph, making them therefore sparse compared to the complete, dense graphs to the left. Because of this sparseness of only optimal TSP edges, it is trivial to decode the optimal TSP solution after the message passing operation of the GNN, given only the colors of the nodes in the updated graph representation (now ignoring the edges in the graph!). We visualize the process in Figure 1.2: W.l.o.g., we start at the node at the bottom which is half white. This node also has a red and orange part. W.l.o.g., we focus on the red part and travel to the node which is half red. This node is also partially white and light blue. We already visited the half-white node, so we go on to the half-light-blue node. This node is also half red and half dark blue. We already visited the half-red node, so we go on to the half-dark-blue node, and so on. In contrast, this decoding strategy would not be possible with the dense graph obtained after message-passing of Figure 1.1, because each node carries information from every other node, so where would we go next in each decoding step?

This idea underlines how a GNN can create more powerful encodings when there is structural information in the graph to exploit. As a consequence, the following questions arise:

- Can we induce structural information on TSP graphs by graph sparsification, enabling neural encoders like GNNs to produce more powerful encodings?
- Which graph sparsification methods can be used and how can the risk of accidentally deleting important edges in the sparsification process be minimized?

1.2 Contributions

To tackle the shortcomings of GNNs on dense graphs, such as graphs representing TSP instances, we provide the following contributions:

- We present the idea of graph sparsification as a form of data preprocessing for instances of the traveling salesman problem. This preprocessing

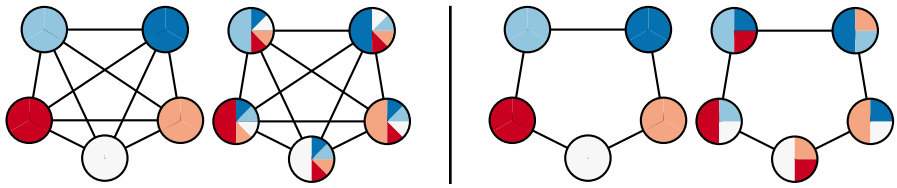


Figure 1.1: Message Passing on Dense (left) and Sparse (right) Graph

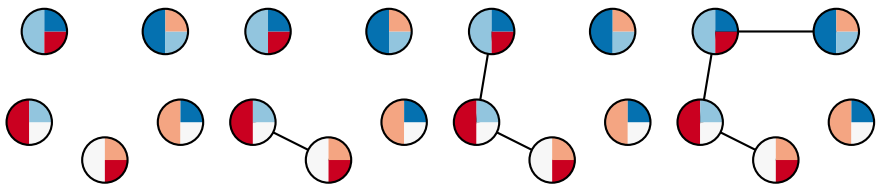


Figure 1.2: Decoding Sparse Graph Encodings

aims to remove unpromising edges (i.e., edges unlikely to be part of the solution) from the TSP instance, allowing the GNN to focus on the more promising parts of the problem.

- We propose two methods for determining the promising edges in the TSP graph.
  1. k-NN is a simple and fast way to preprocess the TSP instance by only keeping the k shortest edges for each node in the graph.
  2. A modified minimum spanning tree (MST) based approach gives additional guarantees for our processed graph such as connectedness.
- We generalize the idea of graph sparsification for GNNs to the concept of attention masking for transformer architectures operating on TSP data.
- We propose ensemble models of different sparsification levels to decrease the risk of optimal edge deletion while still providing additional structure to exploit for the overall model.

1.3 Thesis outline

We provide a background on important concepts necessary for this work in Chapter 2. Afterwards, we give an overview of the current state of learning-based routing concepts in the related work section in Chapter 3. The methodology for sparsifying TSP instances can then be found in Chapter 4. A summary of the paper on which this thesis is based can be found in Chapter 5. The papers is also appended in the second part of this thesis. Chapter 6 concludes the work and gives an overview of potential future research directions.

CHAPTER 2

Background

In this Chapter, we provide the necessary background information for routing problems, used machine learning architectures, and the graph-theoretical concept of minimum spanning trees.

2.1 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a combinatorial optimization problem. Despite its NP-hard nature, the problem is easy to describe: Given a set of cities, a traveling salesman wants to visit all the cities at hand exactly once while ending the journey in the same city where it was started and minimizing the overall traveled distance.

Without loss of generality, we assume there are n cities to be visited, $\{1, \ldots, n\}$. Then, the problem can be represented as a graph problem with a graph $G = (V, E)$ and $V = \{1, \ldots, n\}, E = \{(i, j) | i \neq j, i, j \in V\}$. To determine the solution to the problem, a cost metric is required. This cost metric assigns a weight to each edge in the graph representing the distances between the cities. The TSP can then be solved by finding a Hamiltonian cycle in the graph with minimal weight.

Classically, the problem can also be modeled and solved as an integer linear program, e.g. by the following formulation by Dantzig–Fulkerson–Johnson [3]:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} d_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{i=1, i \neq j}^{n} x_{ij} = 1, && j = 1, \ldots, n \\
& \sum_{j=1, i \neq j}^{n} x_{ij} = 1, && i = 1, \ldots, n \\
& \sum_{i \in S} \sum_{i \neq j j \in S} x_{ij} \leq |S| - 1, && \forall S \subseteq \{1, \ldots, n\}, |S| \geq 2 \\
& x_{ij} \in \{0, 1\}, && i, j = 1, \ldots, n
\end{aligned}
$$

In this formulation, $x_{ij} = 1$ indicates that in the solution one travels from city i to j. d_{ij} reflects the distance between city i and j. The first constraint ensures that each city is entered exactly once whereas the second constraint ensures that each city is left again. The third constraint eliminates the possibility of unconnected subtours. The subtour elimination constraints result in an exponential number of constraints which makes solving the integer linear problem infeasible in practice for big TSP instances with many cities.

Capacitated Vehicle Routing Problem

A simple extension of the TSP is the Capacitated Vehicle Routing Problem (CVRP). The problem can be described as follows: Given a delivery vehicle with a maximum capacity C, we have a special *depot* node where the vehicle starts and ends its tours. Furthermore, we have a set of customer nodes, where each customer i has a demand for goods that occupy a certain capacity c_i (e.g., the weight of the goods). As the delivery vehicle only has the aforementioned limited maximum capacity C, it can potentially not serve all customer demands at once, making it necessary for the vehicle to return to the depot in an intermediate step before serving the next customers. This means that, in contrast to the TSP, there is a special node (the depot) that can be visited multiple times. The overall goal of the problem stays the same, however: we want to minimize the traveled distance. We note that the TSP is

a special case of the CVRP where the vehicle has an infinite capacity, making intermediate returns to the depot unnecessary.

Graph Neural Networks

Graph neural networks (GNNs) are a class of neural architectures where the structure of the neurons reflects the structure of the input. This makes them different from other neural architectures like multi layer perceptrons (MLPs) where the connections between neurons are fixed. GNNs have achieved promising results in many tasks operating on graph-structured data such as molecules, social networks, and traffic models [4]. GNNs iteratively compute feature vector representations for the nodes in the input graph. These representations are updated in each layer of the neural network by performing aggregation operations over the nodes' neighborhoods. Additional learnable weights and activation functions allow the network to process the inputs further. An example of how a node's feature vector is updated in a simple GNN architecture is the following:

$$h_v^{i+1} = \varphi\Big(W^i h_v^i + (\sum_{u \in N(v)} U^i h_u^i) + b^i\Big),$$

where $W^i, U^i,$ and b^i are learnable weights of suitable sizes and ϕ is a non-linear activation function such as sigmoid or ReLU. The feature vectors of node v in layer $i+1$ is then computed by multiplying the previous layers feature vector of the node h_v^i and the feature vectors of the nodes in v's neighborhood $N(v)$ with these learnable weights and aggregating them before applying the non-linearity. The aggregations over the node neighborhoods (which do not have to be summations necessarily but other options such as, e.g., mean operations are also possible) are called *message-passing* operations.

After performing several message passing and update iterations, the node feature vectors of the last network layer can be used for node-level classification or regression tasks. Alternatively, the node feature vectors can also be merged by aggregating or concatenating them and then be used for graph-level tasks.

Noticeable versions of GNNs are the graph convolutional network (GCN; [5]) where messages in the message-passing are weighted by node degrees, giving higher emphasis on nodes with only a few neighbors. Graph attention networks (GAT; [6]) are another popular type of GNNs where the weight of

each message in the message-passing step is determined using the attention mechanism [7].

Limitations of Graph Neural Networks

In terms of their discriminative power to distinguish between different graph-structured inputs, GNNs are known to be related to the Weisfeiler-Leman (WL) graph isomorphism heuristic which has similar aggregation and update operations as GNNs [1], [2]. WL is known to be unable to differentiate between regular graphs (graphs where all nodes have the same degree) [8]. As a result, GNNs cannot distinguish between regular graphs either. This limitation has consequences in real-world datasets. For example, as pointed out in [9], GNNs cannot distinguish between the two molecules decalin and bicyclopentyl. In [10], this limitation has been tackled be explicitly encoding information about structural information not detectable by vanilla GNNs in the initial node feature vectors passed to the GNN. By this, structural information that would otherwise been hidden from the GNN was made accessible to it.

2.2 Transformer Networks

Transformers are a machine learning architecture that recently excelled in natural language processing tasks [7]. However, via graph attention networks, transformers are also closely related to GNNs [11]. In fact, GATs operating on complete graphs are equivalent to transformers. It is possible to apply a concept called *attention masking* to transformers, meaning that the model is unable to consider (or attend to) certain inputs. The concept was already introduced in [7], as in NLP tasks it can be important for the model not to attend to certain inputs (e.g., words in the "future" of a text or padding tokens). However, we can also use attention masking for transformers such that when applied to graph-structured inputs, attending is only possible along the edges of the graph.

2.3 Minimum Spanning Trees

A minimum spanning tree (MST) is a subgraph of an undirected, weighted, connected graph. Weighted means that every edge in the graph has an asso-

ciated cost, like, in the case of the routing setting of this work, the distance between the nodes. A spanning tree of a graph consists of a subset of edges such that all vertices of the graph are still connected and there are no cycles. The minimum spanning tree is such a spanning tree on a weighted graph with the additional property that the sum of the MST edge weights is minimal. Finding the MST of a graph is a combinatorial optimization problem. However, in contrast to the TSP, the problem can be solved in polynomial time, e.g., with the well-known algorithms of Prim [12] or Kruskal [13]. We provide an example of an MST in Figure 2.1, where the green edges are part of the MST and the light-grey edges are part of the underlying original graph.

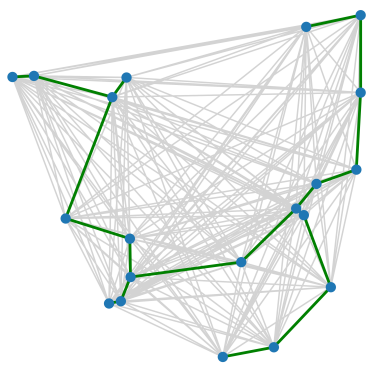


Figure 2.1: A Minimum Spanning Tree

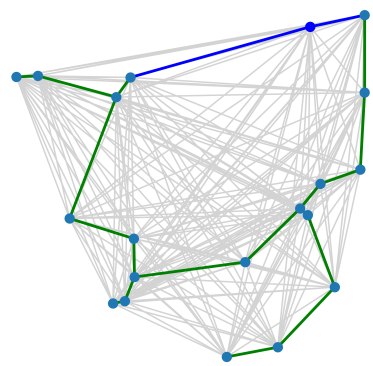


Figure 2.2: A 1-Tree

1-Trees

1-Trees are graph structures related to MSTs. Despite their name, 1-Trees are not trees, since they contain cycles. A 1-Tree can be created from a graph $G = (V, E)$ (w.l.o.g., we assume $V = \{1, \ldots, n\}$) by finding a MST on the set of edges $V \setminus \{1\}$ where 1 is an arbitrary node of V. After finding an MST on $V \setminus \{1\}$, a 1-Tree is created by combining the set of edges of the MST with two edges from E that are incident to 1. We provide an example of a 1-Tree in Figure 2.2, where the dark blue node has been chosen arbitrarily and we further chose two arbitrary edges to connect this special node with

the MST computed on the remainder of the graph. We note that there also exist *minimum* 1-Trees which are 1-Trees with minimal weights. Minimum 1-Trees are interesting in our routing setting because their cost form a lower bound on the cost of an optimal TSP tour. As a result, minimum 1-Trees play an important role in the popular LKH algorithm [14].

CHAPTER 3

Related Work

We give an overview of existing work dealing with learning-based routing problem solvers in this section. We categorize these works by the way learning is used within the overall solver frameworks. Further, we provide a short review of learning-based studies that deal with the concept of sparsification of the TSP.

3.1 Learning to Route

With the rise of deep learning in the last few years, there have also been many proposals to develop machine learning-based frameworks for solving routing problems. The different frameworks use machine learning in many different ways. Furthermore, the suggested frameworks use a variety of neural network architectures and learning paradigms (supervised learning, reinforcement learning or unsupervised learning). In this work, we try to classify the papers by the way machine learning is incorporated into the framework. In general, papers use typically one of two approaches in their machine learning-based framework: In the first approach category, solutions for the routing problem are constructed from scratch. Therefore, in the following, we refer

to these approaches as *construction-based*. In the second approach category, solutions are iteratively improved. This means, an initial solution is passed to the neural framework and it iteratively applies improvement operations until a convergence criterion is met. Hereinafter, we refer to these approaches as *improvement-based*. The proposed categories can be further split into subcategories. Construction-based approaches can be *one-shot-approaches* or *incremental-approaches*, depending on whether the machine learning framework is only applied once or several times to incrementally build a solution. Improvement-based approaches can be *heuristic-based* in case they learn improvement operators known from classical heuristics such as k-opt. However, they can also be *subproblem-based* when the framework iteratively selects and improves subproblems to generate overall better solutions in a divide-and-conquer manner. We note that the categories mentioned so far capture the majority of papers that try to use machine learning to tackle routing problems. However, there are further possibilities, e.g., by using machine learning to facilitate decisions in traditional solvers (like branching in branch-and-bound algorithms) or to restrict the search space. In the following, we provide an overview of the approaches mentioned so far.

Construction-based Approaches

Incremental-based Approaches

In incremental-based approaches, solutions are built by iteratively adding one city at a time to a partial solution until they form a valid tour. The machine learning model's task is, typically, to autoregressively select the next city to visit. Let's consider an example of a small TSP instance visualized in Figure 3.1. Starting from node 3, in the first step node 1 is selected as the next one to visit. Afterwards, the connection (1,4) is chosen. Then, we travel to node 5. To complete the tour, we would probably travel to node 2 next and finally back to node 3.

A neural framework constructing solutions for routing problems this way typically consists of two parts: An encoder and a decoder architecture. The encoder captures the instance by encoding all the information of individual nodes in high-dimensional feature vectors. These encodings typically embed information of the node coordinates in a coordinate frame, the distances between each other, and additional information like, e.g. in the case of the

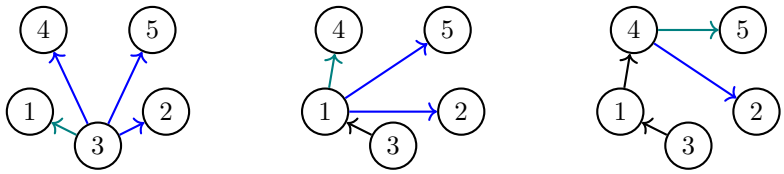


Figure 3.1: Incremental Solution Construction

CVRP, capacity demands customers might have. Then, the partial solution can be represented as a combination of the high-dimensional feature vectors and the decoder iteratively predicts probabilities of selecting new nodes to add to the solution. These probabilities can be used for sampling, greedy decoding, or more advanced search algorithms (e.g., Monte Carlo tree search (MCTS; [15]) or beam search [16]). We note that typically probabilities are masked in a way to ensure valid solutions, e.g., in a way such that nodes cannot be visited multiple times, or, in the case of CVRP, capacity constraints are respected.

Examples of such incremental approaches are the works of [17]–[21]. All of these papers follow the RL paradigm. Neural architecture-wise, [18] (who solve the CVRP in their framework) relies on a recurrent neural network (RNN) whereas the other works rely on Transformer-based architectures. [17] designs a framework for TSP, while [19] generalizes to TSP, CVRP and other routing problem variants. [20] introduces a shared baseline in the RL framework by performing multiple solution rollouts, enhancing the performance of existing frameworks. [21] generalizes to bigger TSP instances with up to 500 nodes by using reversible residual network layers. There are also papers using SL for incremental-based approaches, e.g. [22] where the authors train a GNN to predict the probability of selecting the next node in the next decoding step. They use these probabilities in a Monte Carlo tree search to find good solutions.

One-shot Approaches

In contrast to incremental approaches, where probabilities where a model is applied over and over again to predict probabilities for the next node to visit, in one-shot approaches, these probabilities are generated at once.

Let us consider the same small TSP instance from before, now represented

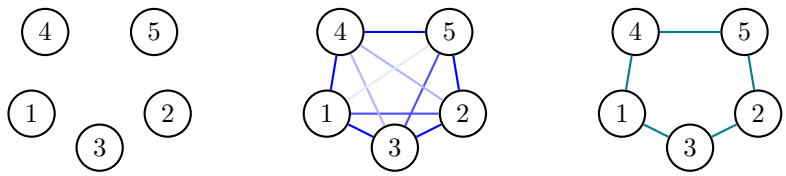


Figure 3.2: One-shot Solution Construction

in Figure 3.2. To the very left, we have the representation of the nodes in a coordinate frame, still without any connections between them. After the machine model is applied, probabilities are predicted indicating how likely it is to travel from one node to another in the solution tour. In the figure, darker connections represent higher probabilities. Finally, a search algorithm or some other decoding technique transforms these probabilities in a valid solution (to the very right).

Within the ML framework, the pairwise probabilities to travel from one node to another are typically predicted in the form of probability heatmaps represented as matrices. Examples for such approaches are [23]–[27]. [23] trains GNN in an SL setting to predict probability heatmaps for TSP which are decoded to valid solutions by beam search. [24] generalizes the idea to big TSP instances with thousands of nodes by sampling subgraphs and merging their heatmaps. The solutions are then created by MCTS from the heatmaps. [25] uses dynamic programming to decode the heatmaps and further generalizes the idea to CVRP. [26] gets rid of the supervision requirement by designing an RL framework to generate heatmaps for TSP. [27] developed a completely unsupervised framework for TSP by using a surrogate loss, resulting in fast and efficient learning and generalizing to instances with up to 1000 nodes.

Improvement-based Approaches

Heuristic-based Approaches

In heuristic-based approaches, improvement operators like k-opt are learned. When applying such operators, an initial valid solution is improved over and over until convergence (or some termination criterion is met). As the final output is not necessarily an optimal solution, we refer to such algorithms as heuristics. Examples of traditional (i.e., non-ML-based) algorithms using

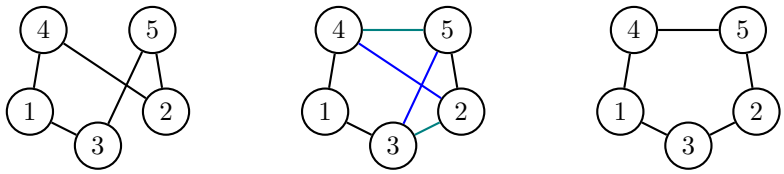


Figure 3.3: 2-opt move

such improvement operators for solving routing problems are LK [28] and its extension LKH3 [14]. Applying k-opt to a routing problem such as TSP means deleting k edges in the current solution and substituting them with k new edges while still ensuring the validity of the new solution. We present an example of a 2-opt move in Figure 3.3. In the 2-opt move, the blue edges {2,4} and {3,5} are deleted and the edges {4,5} and {2,3} are added instead. Note that the node permutation in the old solution was (3,1,4,2,5,3) and it is (3,1,4,5,2,3) in the new solution (3 is chosen as an arbitrary start node in this representation). We highlight that the edge {2,5} is traversed in the opposite order in the new solution.

Learning such improvement operations typically means learning to select the edges (or the nodes that form the edges) for the swapping operations.

[29] proposes a long short-term memory (LSTM; [30])-based deep Q-learning framework that is applicable to CVRP (among other problems). The framework is trained to pick two nodes in the CVRP solution and the first node is moved after the second one in the new solution. [31] learns 2-opt moves for the TSP in a GNN-based framework with RL. The idea is later generalized to CVRP [32]. Within a transformer-based framework, [33] tries to learn other improvement operations than 2-opt as well such as swapping and relocating nodes for TSP and CVRP. [34] proposes another transformer-based framework trained with RL which can freely select from a set of different improvement operators to solve the CVRP. By this, the authors create the first ML-based framework to outperform the state-of-the-art LKH3 [14] algorithm on CVRP.

Subproblem-based Approaches

Subproblem-based approaches are especially suitable for big problem instances. By selecting and optimizing a subregion of the current valid solution, the overall solution improves in quality until convergence.

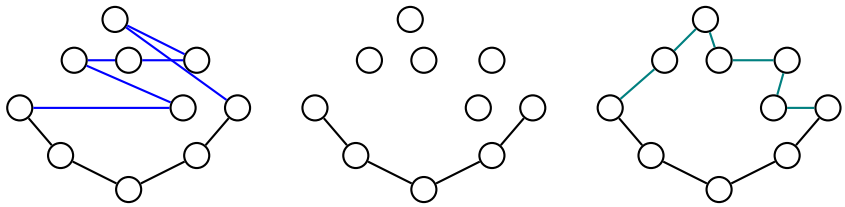


Figure 3.4: Subproblem-based Approaches

In Figure 3.4, we visualize a subproblem-based improvement for a TSP instance. The nodes on the blue path are selected as the subproblem to be improved and deleted. Afterwards, while remembering the end nodes of the path, the shortest path visiting all currently unconnected nodes is constructed and plugged into the solution. An example of such an approach for TSP is [35] where subproblems from the current solution are sampled and reconstructed by a transformer-based model, trained to find the shorted path with RL.

[36] tackles the CVRP by using a transformer-based model trained with supervised learning to predict a possible improvement in subproblems. The most promising subproblem is then optimized by using the LKH3 [14] algorithm.

Similarly, [37] trains an LSTM-based framework with RL to provide CVRP subproblems for the optimization. The optimization of the subproblems is then done by traditional heuristics or it can again be learning-based.

3.2 Sparsifying the TSP

Not many papers using ML to solve routing problems have considered sparsifying the graph representations of TSP instances before passing them to neural encoder architecture such as GNNs or transformers.

[38] sparsifies TSP instances by applying the k-nearest neighbors heuristic to the graph representations before passing them to a GNN encoder in their one-shot-based approach. They did this to reduce the runtime of their framework from $\mathcal{O}(n^2)$ to $\mathcal{O}(kn)$. [39], [40] also use k-nn to overcome the quadratic growth of the number of edges in the TSP graphs used in their learning-based frameworks and, therefore, to reduce the runtime of their frameworks. [41] proposes a GNN to predict scores for the edges in a TSP graphs, afterwards used in a “neural” version of the LKH algorithm and acknowledges the impor-

tance of prior sparsification. They sparsify the graphs by applying 20-nn but do not further investigate the concept of sparsification. [25] do not sparsify the graph passed to the GNN outputting the heatmaps but adjust the heatmaps before passing it to the dynamic programming decoding. This heatmap adjustment is performed in a way such that edges not part of the sparse graph representation are ruled out to be part of the solution generated by the DP. [42] do not use deep learning but SVMs to predict whether edges in a TSP graph are promising to be part of the optimal solution. Unpromising edges are removed to reduce the search space for classical solvers.

We note that none of these papers analyze why sparsification is important (for encoders like GNNs) or provide an extensive study on it. We further note that sparsification in these papers is done to reduce runtime, not to improve solution quality by allowing the encoders to take advantage of the additional structure induced on the TSP graphs. Moreover, most papers use the simple k-nn heuristic as a sparsification strategy and do not consider more sophisticated methods.

CHAPTER 4

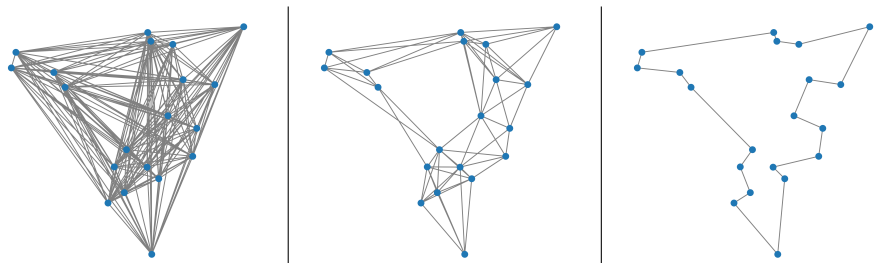
Methodology - Sparsifying the TSP

In this Chapter, we discuss how TSP instances as a representative for routing problems can be made sparse. We note that it is possible to generalize this idea to other routing problems. We further point out that the idea of sparsification is independent of the way learning is incorporated into a deep-learning-based routing framework. If the framework includes an encoder architecture (which virtually all frameworks do), it is possible to adapt it to incorporate the idea of sparsification. The idea is also independent of the learning paradigm (supervised, unsupervised, or reinforcement learning) that is used to train the encoder within the overall framework.

4.1 Sparsification Methods

As outlined in this work so far, GNNs are not suitable to operate on dense graphs such as TSP instances. Therefore, we want to investigate imposing additional exploitable graph structure on the TSP instances through graph sparsification. The goal of this sparsification process is to delete unpromising edges (i.e., edges that are not likely to be part of the TSP solution) from the graph, before applying the GNN encoder.

As the cost of traveling from one node to another in a routing problem is typically equivalent to the distance between these nodes, a straightforward idea is to just include the edges between a node and its k-nearest neighbors, as it has already been done in [38]–[40]. This approach is simple and, additionally, computationally cheap and therefore fast. We visualize the idea of sparsification with k-nn in Figure 4.1. To the left, we show the dense TSP graph. In the center, each node is only connected to its 5 nearest neighbors. To the right, for comparison, we can see the optimal TSP tour.



(a) TSP instance as a complete graph

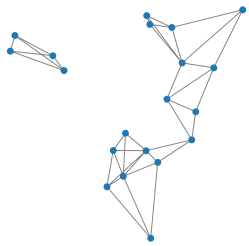
(b) TSP instance with 5-nn edges only

(c) Optimal tour of TSP instance

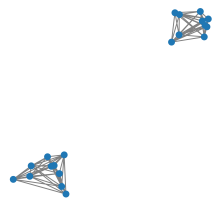
Figure 4.1: A TSP instance with 20 cities

Even though the result of the sparsification in the particular instance Figure 4.1 was good, as it reduced the number of edges in the graph considerably while still including all edges of the optimal tour, it has a serious drawback: The resulting sparse graph can generally be *disconnected*, meaning that it consists of several connected components. This problem is especially prone to occur when a very low k is chosen (meaning we want to delete many edges) or the nodes in the TSP instance are clustered. In fact, if we consider a graph with n nodes and two very dense clusters with $n/2$ nodes each, no $k < n/2$ will suffice to produce a connected sparse graph with k-nn. We give an example of such a clustered graph in Figure 4.2. In this figure, we also show the graph of Figure 4.1 again, now sparsified with 3-nn instead of 5-nn, resulting in an unconnected graph.

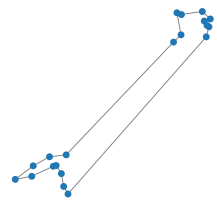
Choosing high k to minimize the risk of unconnected graphs does not seem optimal, as it results in graphs that are still rather dense, reducing the disadvantages of GNNs on dense graph data only unsatisfactorily. However, unconnected graphs are also not acceptable in our setting, since no message-passing



(a) TSP instance sparsified with 3-nn



(b) Clustered TSP instance sparsified with 9-nn



(c) Optimal tour of clustered TSP instance

Figure 4.2: Unconnected sparse TSP instances produced with k-nn

will be performed between the disconnected components of a graph, resulting in encodings that do not carry information on all important connections in the graph.

Therefore, we propose another method of graph sparsification which is based on minimal 1-Trees. MSTs (and therefore (minimal) 1-Trees) are connected by definition, meaning a sparsification approach building upon 1-Trees eliminates the risk of unconnected sparse graphs.

For the sake of simplicity, we explain the idea based on MSTs instead of 1-Trees, but the concept is exactly the same: Given a graph $G = (V, E)$ we can construct its MST T^* which has a unique cost $c(T^*)$. To determine how promising an edge (u, v) is to keep in the sparsified graph, we can construct another spanning tree $T^*_{u,v}$ which should also have minimal cost but at the same time enforce edge (u, v) to be part of it. We can then assign a hypothetical cost $\alpha(u, v)$ to edge (u, v) by computing the difference $c(T^*) - c(T^*_{u,v})$. Note that this cost is non-negative (it is zero exactly for the edges part of T^*) and expresses how much more expensive an MST becomes if edge (u, v) is enforced to be part of it. We visualize the idea in Figure 4.3 where we show the plain MST to the left and the MST enforcing edge (1,2) to the right. $\alpha(1, 2) = 2 - 1.06 = 0.94$, where 2 is the cost of edge $(1, 2)$ and 1.06 is the cost of the no long required edge $(3, 6)$. By computing α for all edges in the graph, we can obtain a sparse TSP instance by the following way: For each node u in the TSP instance, we keep the k edges $(u, v), v \in V \setminus \{u\}$ with the lowest α. We also describe the procedure in Listing 4.1. This idea (based on

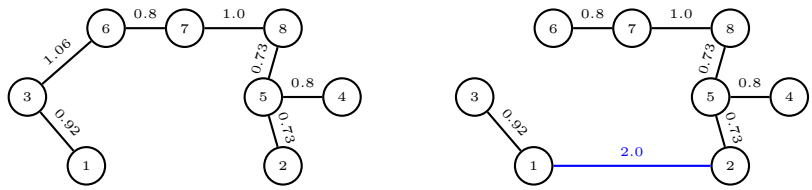


Figure 4.3: MST (left) and the MST with enforced edge (1,2)

1-Trees) is also used in the LKH algorithm [14] to find candidate edges for k-opt moves in the improvement framework of this algorithm. In the LKH algorithm, the procedure is further refined by applying a gradient-optimization method to adjust the weights in the original graphs and by this obtaining minimal 1-Trees which are closer to the optimal TSP solution.

We want to emphasize here that the sparsification procedure is not only beneficial to GNN but can also be applied to transformer networks in the form of attention masking. There, the attention masks can easily be computed from the adjacency matrices of the sparse graphs. If the adjacency matrix of the sparse graph indicates an edge (i.e., the entry of the matrix is 1), we do not mask the attention score at the corresponding position in the transformer network. Otherwise, the attention is masked, disabling the transformer to attend along this connection when computing the node embeddings.

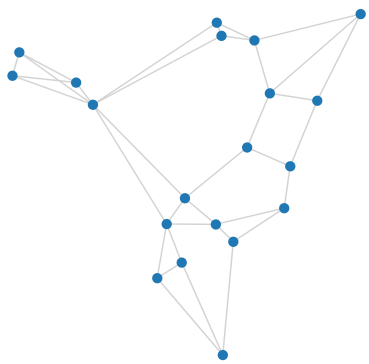


Figure 4.4: The graph from Figure 4.2a, now sparsified with the MST-based approach, keeping the 3 most promising edges for each node.

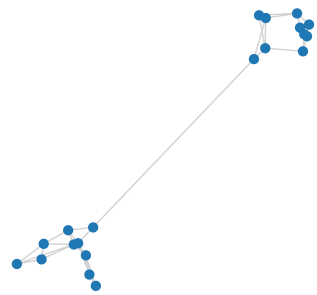


Figure 4.5: The clustered graph from Figure 4.2b, now sparsified with the MST-based approach, keeping the 3 most promising edges for each node.

```python
def mst_sparsification(G: Graph, c: CostMetric, k:int):
    """
    Returns a sparse graph given a dense TSP graph.

    Parameters:
    G = (V, E) Graph: The dense TSP graph.
    c CostMetric: A cost metric assigning a cost to each edge in G.
    k int: Determines how many edges shall be kept for each node in G.

    Returns:
    Graph, a sparsified graph (V, sparse_edges)
    """
    T_opt = MST(G, c) # get the MST
    alpha = dict() # init the alpha dict

    for u in V:
        for v in V:
            if u == v:
                continue
            T_opt_u_v = MST(G, c, u, v) # get the MST enforcing edge (u,v)
            alpha[(u,v)] = cost(T_opt_u_v) - cost(T_opt) # compute the alpha value

    sparse_edges = set() # init the sparse graph edges
    for u in V:
        edge_list = list() # a list of all edges with endpoint u
        for v in V:
            if u == v:
                continue
            edge_list.append((alpha[(u,v)], (u,v))) # tuples of form (alpha, edge)
        edge_list.sort(key=lambda x: x[0]) # sort based on alpha
        edge_list = [edge for (alpha, edge) in edge_list] # keep only edge
        sparse_edges.update(edge_list[:k]) # add k best edges to sparse_edges

    return (V, sparse_edges)
```

Listing 4.1: MST-based Sparsification

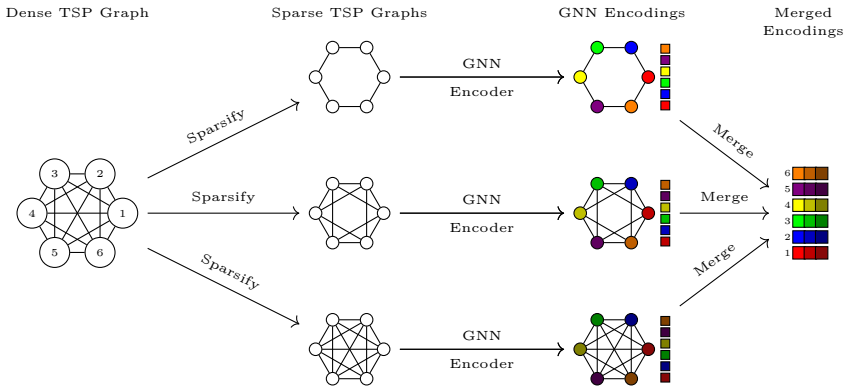


Figure 4.6: Ensemble Encoder Using Different Sparsification Levels

4.2 Sparse Graph Ensembles

So far, we introduced graph sparsification as a procedure of TSP instance preprocessing which allows GNN encoders to create more expressive encodings. However, we also noted that simple heuristics like k-nn carry the risk of producing disconnected sparse graphs. We overcome this limitation by proposing an MST-based sparsification concept. However, the MST-based sparsification method also cannot guarantee never to delete edges from the optimal TSP tour when producing the sparse graph. This might result in node embeddings where two nodes, that should be visited after each other in the optimal TSP tour, do not carry information about each other in their embedding representations.

As a trade-off between sparse graphs where each node is only connected to its most promising neighbors and dense graphs where the risk of optimal edge deletion is minimized, we propose sparse graph ensembles. The idea is that each TSP instance is encoded several times, with each encoding being performed on a TSP graph representation of different sparsification levels. After the encodings are generated for each sparsification level, they are merged on a node level. This means, that the different encodings for a single node in the different graphs are merged by, e.g., concatenation or averaging. By this, a single encoding for each node is obtained again which can be processed in existing ML-based routing frameworks as usual.

```python
def get_mst_ensemble_encoding(G: Graph, c: CostMetric, k_values):
    """
    Returns an ensemble encoding of different MST-based sparsification of G

    Parameters:
    G = (V, E) Graph: The dense TSP graph.
    c CostMetric: A cost metric assigning a cost to each edge in G.
    k_values list: A list of k values used in the sparsification processes

    Returns:
    An encoding of the graph based on different sparsification levels
    """
    N = len(V) # number of nodes
    ensemble_encodings = list() # a list to save the encodings
    for _ in range(N): # initialize the encoding for each node as an empty list
        ensemble_encodings.append(list())

    for k in k_values: # iterate over different sparsification levels
        sparse_graph = mst_sparsification(G, c, k) # get the sparse graph
        individual_encoding = get_graph_encoding(sparse_graph, k) # encodings are of size N x hidden_dim
        for i in range(N): # for each node, add the encodings of current sparsification level
            ensemble_encodings[i].extend(individual_encoding[i])

    return ensemble_encodings
```

Listing 4.2: Ensemble Encoder

We describe the procedure in Listing 4.2. Further, we visualize the idea in Figure 4.6. In the figure, a TSP instance with six nodes is given. In the beginning, it is represented as a dense, complete graph. In the first step, the instance is sparsified several times with different sparsification levels. In the example at hand, possibly 2-nn, 4-nn, and 5-nn were applied (the latter sparsification resulted in a dense graph again because of the small size of the example). Afterwards, a GNN computed encodings for all graph representations, we visualize this by assigning different colors to the nodes. Furthermore, we note that we can extract the node embeddings from the graph, represented

as the column of different node colors next to it. We note that these colors are ordered by using the node order depicted in the initial dense graph (w.l.o.g. (6,5,4,3,2,1)). In the last step, the different embeddings computed for each node are merged. E.g., the three encodings for node 1 (which are all reddish) are merged. The merging in the figure is done by concatenation, which could be followed by an MLP to reduce the dimensionality of the encodings or an alternative averaging.

We note that it is also possible to generate transformer-based ensemble encoders where different attention masks are produced from the different sparse graphs.

CHAPTER 5

Summary of included papers

This chapter provides a summary of the included papers.

5.1 Paper A

Attila Lischka, Jiaming Wu, Rafael Basso, Morteza Haghir Chehreghani, Balázs Kulcsár
Less Is More – On the Importance of Sparsification for Transformers and Graph Neural Networks for TSP
Submitted to *IEEE Transactions on Neural Networks and Learning Systems* in March 2024 .

In this paper, the concept of graph sparsification for GNN encoders on TSP data was introduced. Two sparsification methods were proposed and their performance in keeping optimal TSP edges in sparsified graphs was evaluated. This evaluation was performed for different sparsification levels and on different distributions of TSP nodes in the coordinate frame. The first sparsification method was k-nearest neighbors where the edges (i,j) to the k closest nodes j in the TSP graph are kept for each node i. This spar-

sification method has the advantage of being fast, while on the other hand having a high risk of deleting optimal edges or producing disconnected sparse graphs. The second sparsification method was based on 1-Trees, a variant of minimum spanning trees, which is computationally more expensive but less likely to delete optimal edges. Furthermore, this approach is guaranteed to produce connected sparse graphs. Afterwards, an incremental-construction-based framework was adapted to incorporate different types of GNNs (Graph Attention Networks and Graph Convolutional Networks) operating on sparse data. By this, it was shown that the performance of the overall architecture increases if the GNN encoders operate on sparse TSP data. This performance increase was up to a factor of $\times 22$, depending on the exact GNN architecture and the data distribution. The idea was further generalized to transformer encoders by leveraging attention masking. Here, the attention masks imposed on the transformers reflect the adjacency matrices of sparsified TSP graphs. Moreover, ensemble encoders of different sparsification levels were adopted. The resulting transformer-based encoder ensembles achieved state-of-the-art performance within the domain of incremental-construction-based frameworks with optimality gaps as good as 0.10% on TSP with 100 nodes.

CHAPTER 6

Concluding Remarks and Future Work

6.1 Conclusion

Routing problems such as the TSP are a class of NP-hard combinatorial optimization problems that have been tackled using machine learning-based frameworks in recent years. In this work, we explored the sparsification of data instances when using GNN encoders in such learning-based frameworks to solve routing problems. We discussed two sparsification methods, the simple k-nearest neighbor heuristic and a procedure based on minimum spanning trees with the first one being simple and computationally cheap and the second one guaranteeing connected sparse graphs. We generalized the idea of graph sparsification for GNN encoders to transformer-based encoders by applying attention masking. Furthermore, we presented an ensemble method where a TSP instance is sparsified several times with different sparsification levels. Afterwards, encodings for all sparsified graphs are computed which are merged in the end. By this, it is possible to provide guidance for the encoder on what the most promising edges in a graph are while still minimizing the risk of completely deleting optimal edges in the sparsification process.

6.2 Future Work

So far, our encoders operating on sparse data have only been tested in a construction-based framework. As pointed out in this work, the idea is highly flexible, however, and can easily adapted to compute encodings in one-shot-based, heuristic-based, or subproblem-based approaches as well.

Furthermore, the idea of sparsifying data instances for routing problems has only been adapted for TSP. However, there are many extensions of the TSP that are also tackled by learning-based solvers that could benefit from sparsification too. Examples are the traveling salesman problem with time windows (TSPTW), the Prize Collecting TSP (PCTSP), or the capacitated vehicle routing problem. The latter two were, e.g., tackled by the learning-based framework in [19].

Similarly, the sparsification idea can be deployed to more applied settings. The TSP is a rather theoretical problem dealt with by many papers in the computer science community. However, it would be interesting to adapt the sparsification framework in more applied settings where problems with real-world data and challenges are tackled. As an example, we mention [43] where the goal was to route an electric vehicle, plan the charging, and prevent potential battery depletion.

Another open question is if it is possible to provide theoretical guarantees or bounds for the sparsification procedures. This means predicting a value on how many optimal edges are expected to be deleted on accident when a specific sparsification method and sparsification level are applied. Furthermore, other sparsification methods can be developed and tested, possibly with a view to more constrained routing problems (like the aforementioned PCTSP or TSPTW) and special data distributions (e.g., clustered data).

Overall, possible future directions are:

- Implementation of encoders leveraging sparsification in one-shot-based, heuristic-based, or subproblem-based learning frameworks.
- Adapting sparsification to CVRP, PCTSP, TSPTW, and similar routing problems.
- Deployment of sparsification-based encoders to real-world, applied settings.
- Development of new sparsification methods, (potentially considering dif-

ferent routing problems and data distributions) as well as theoretical analysis.

References

[1] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" In *International Conference on Learning Representations*, 2019.

[2] C. Morris, M. Ritzert, M. Fey, *et al.*, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 4602–4609.

[3] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the operations research society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[4] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[6] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, *et al.*, "Graph attention networks," *stat*, vol. 1050, no. 20, pp. 10–48 550, 2017.

[7] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[8] S. Kiefer, "Power and limits of the weisfeiler-leman algorithm," Ph.D. dissertation, Dissertation, RWTH Aachen University, 2020, 2020.

[9] R. Sato, "A survey on the expressive power of graph neural networks," *arXiv preprint arXiv:2003.04078*, 2020.

[10] G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein, "Improving graph neural network expressivity via subgraph isomorphism counting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 1, pp. 657–668, 2022.

[11] C. Joshi, "Transformers are graph neural networks," *The Gradient*, 2020.

[12] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[13] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[14] K. Helsgaun, "An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems: Technical report," 2017.

[15] C. B. Browne, E. Powley, D. Whitehouse, *et al.*, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[16] P. S. Ow and T. E. Morton, "Filtered beam search in scheduling," *The International Journal Of Production Research*, vol. 26, no. 1, pp. 35–62, 1988.

[17] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, "Learning heuristics for the tsp by policy gradient," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings 15*, Springer, 2018, pp. 170–181.

[18] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," *Advances in neural information processing systems*, vol. 31, 2018.

[19] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" In *International Conference on Learning Representations*, 2019.

[20] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "Pomo: Policy optimization with multiple optima for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 188–21 198, 2020.

[21] Y. Jin, Y. Ding, X. Pan, *et al.*, "Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 8132–8140.

[22] Z. Xing and S. Tu, "A graph neural network assisted monte carlo tree search approach to traveling salesman problem," *Ieee Access*, vol. 8, pp. 108 418–108 428, 2020.

[23] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv preprint arXiv:1906.01227*, 2019.

[24] Z.-H. Fu, K.-B. Qiu, and H. Zha, "Generalize a small pre-trained model to arbitrarily large tsp instances," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 7474–7482.

[25] W. Kool, H. van Hoof, J. Gromicho, and M. Welling, "Deep policy dynamic programming for vehicle routing problems," in *International conference on integration of constraint programming, artificial intelligence, and operations research*, Springer, 2022, pp. 190–213.

[26] Y. L. Goh, W. S. Lee, X. Bresson, T. Laurent, and N. Lim, "Combining reinforcement learning and optimal transport for the traveling salesman problem," *arXiv preprint arXiv:2203.00903*, 2022.

[27] Y. Min, Y. Bai, and C. P. Gomes, "Unsupervised learning for solving the travelling salesman problem," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[28] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Oper. Res.*, vol. 21, pp. 498–516, 1973.

[29] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," *Advances in neural information processing systems*, vol. 32, 2019.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[31] P. R. d O Costa, J. Rhuggenaath, Y. Zhang, and A. Akcay, "Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning," in *Asian conference on machine learning*, PMLR, 2020, pp. 465–480.

[32] P. da Costa, J. Rhuggenaath, Y. Zhang, A. Akcay, and U. Kaymak, "Learning 2-opt heuristics for routing problems via deep reinforcement learning," *SN Computer Science*, vol. 2, pp. 1–16, 2021.

[33] 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, 2021.

[34] H. Lu, X. Zhang, and S. Yang, "A learning-based iterative method for solving vehicle routing problems," in *International conference on learning representations*, 2020.

[35] H. Cheng, H. Zheng, Y. Cong, W. Jiang, and S. Pu, "Select and optimize: Learning to aolve large-scale tsp instances," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2023, pp. 1219–1231.

[36] S. Li, Z. Yan, and C. Wu, "Learning to delegate for large-scale vehicle routing," *Advances in Neural Information Processing Systems*, vol. 34, pp. 26 198–26 211, 2021.

[37] Z. Zong, H. Wang, J. Wang, M. Zheng, and Y. Li, "Rbg: Hierarchically solving large-scale routing problems in logistic systems via reinforcement learning," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '22, Washington DC, USA: Association for Computing Machinery, 2022, pp. 4648–4658, ISBN: 9781450393850.

[38] R. Qiu, Z. Sun, and Y. Yang, "Dimes: A differentiable meta solver for combinatorial optimization problems," *Advances in Neural Information Processing Systems*, vol. 35, pp. 25 531–25 546, 2022.

[39] Z. Sun and Y. Yang, "Difusco: Graph-based diffusion solvers for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 36, pp. 3706–3731, 2023.

[40] H. Ye, J. Wang, Z. Cao, H. Liang, and Y. Li, "Deepaco: Neural-enhanced ant systems for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

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

[42] Y. Sun, A. Ernst, X. Li, and J. Weiner, "Generalization of machine learning for problem reduction: A case study on travelling salesman problems," *OR Spectrum*, vol. 43, pp. 607–633, 2021.

[43] R. Basso, B. Kulcsár, B. Egardt, P. Lindroth, and I. Sanchez-Diaz, "Energy consumption estimation integrated into the electric vehicle routing problem," *Transportation Research Part D: Transport and Environment*, vol. 69, pp. 141–167, 2019, ISSN: 1361-9209.