ML-based Combinatorial Optimization for Solving the Realistic Vehicle Routing Problem

Authors:
Ashish Kumar Sahu 311880
Ilayda Topuz 312378
Jaishree Janu 311119
Jatin Garg 311226
Sophia Lawal 310882

Supervisors:
Daniela Thyssens
Dr. Dr. Lars Schmidt-Thieme

31st March 2023

Student Research Project 2022-2023
Master of Science in Data Analytics

Wirtschaftsinformatik und Maschinelles Lernen
Stiftung Universität Hildesheim
Universitatsplätz 1, 31141 Hildesheim

Statement as to the sole authorship of the paper:

"ML-based Combinatorial Optimization for Solving the Realistic Vehicle Routing Problem." We certify that we solely wrote the report named above and that no assistance was used other than that cited. The passages in this paper that were taken verbatim or with the same sense as that of other works have been identified in each case by the citation of the source or the origin, including the secondary sources used. This also applies to drawings. Sketches, illustrations, internet sources, and other electronic texts or data collections, etc. The submitted paper has not been previously used to fulfill degree requirements nor published in English or any other language. We are aware of the fact that false declarations will be treated as fraud.

31.03.2023, Hildesheim

Ashish Kumar Sahu

Ilayda Topuz

Jaishree Janu

Jatin Garg

Sophia Lawal

Abstract

Due to urbanization on a global scale, big challenges in transportation systems, logistics operators, and e-commerce enterprises have emerged with the rising urban population. These challenges are mathematically formulated as Vehicle Routing Problems (VRPs), known as NP-hard. It is still a research question to solve these problems efficiently and in cases when the problem size is large. Although many heuristic approaches and machine learning (ML) based approaches exist to solve them, most of these methods lack a realistic setting. For instance, most of them assume Euclidean distances in the graph, which are symmetric and non-realistic. There are also some works that obtain real-distance data which are not Euclidean; however, they still neglect that in real-world scenarios, distances are not static; they may have deviations due to several factors, such as traffic, weather conditions and construction work. In this paper, we propose a more realistic data generation procedure in which we apply realistic distortions in the real location and distance data we have obtained to reflect the real-time variations in the distances. Then, we adopt a Deep Reinforcement Learning (DRL) based approach to solve this problem in a dynamic setting. Our policy is based on a Graph Attention Network that takes graph states as input and generates the optimal solution autoregressively. Finally, we performed multiple experiments on static as well as dynamic distance datasets with varying graph sizes, and we compared our model with the baselines. Our work can be followed and viewed on the GitHub link: `https://github.com/JaishreeJanu/realistic_vrp_drl`

Contents

List of Figures

List of Tables

List of Abbreviations

- VRP - Vehicle Routing Problem
- CVRP - Capacitated Vehicle Routing Problem
- OR - Operations Research
- ML - Machine Learning
- RL - Reinforcement Learning
- DRL - Deep Reinforcement Learning
- TSP - Travelling salesman problem
- TS - Tabu Search
- SA - Simulated Annealing
- GRASP - Greedy Randomized Adaptive Search Procedure
- GA - Genetic Algorithm
- ACO - Ant Colony Optimization
- GCN - Graph Convolution Network
- PN - Pointer Network
- GNN - Graph Neural Network
- RNN - Recurrent Neural Network
- MHA - Multi-Head Attention
- RGCMA - Residual Graph Convolutional encoder and Multiple Attention-based decoders

Chapter 1

Introduction

The recent emergence of urbanization on a global scale is an irrefutable fact, as many people move toward developed cities in search of life-elevating opportunities. Such a rising urban population brings big challenges to the social and industrial landscapes that require high mobility to thrive - transportation systems, logistics operators, e-commerce enterprises, smart city projects, etc. Vehicle Routing Problems (VRPs) are a fantastic mathematical formulation of the situation and are the most prominently studied family of combinatorial optimization problems of all times [22]. The Capacitated Vehicle Routing Problem (CVRP) is a well-known variant of the VRP that is applied extensively to algorithmically generate optimal tour solutions for modern-day urban operation systems, leading to substantial cost savings. A verbal definition of the underlying CVRP can be stated as follows:

Given: (a) A set of logistical demands from customers spread across various locations in a region, and (b) a homogeneous fleet of vehicles, each having a maximum goods carrying capacity to service these customers.

Task: Generate a set of vehicle routes to fulfill the customers' demand with minimum transportation cost. The generated tour solutions should be near-optimal and feasible so each customer is visited only once. The time window constraints are not considered.

Imagine that you are an e-commerce supplier who wants to deliver a number of parcels to customers in your city with minimum transportation costs or a professional attending an intra-day business conference whose purpose is to gain maximum relevant information within the given time frame. The applicability of the CVRP in various real-world scenarios is evident, which further establishes its importance in our day-to-day lives. Generating optimal transport solutions significantly aids organizations in the form of

reduced costs, increased customer satisfaction, and hence higher profit numbers. Unfortunately, this combinatorial optimization problem is NP-hard to solve optimally, i.e., it cannot be solved in polynomial time [25]. This is due to an exponential increase in the solution search space as customer nodes are upscaled.

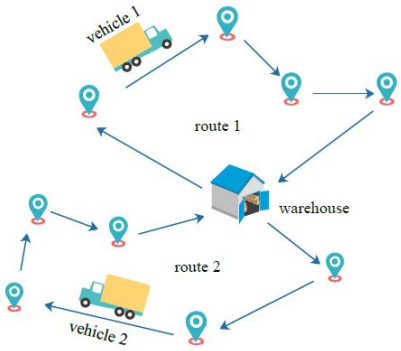


Figure 1.1: Illustration of a VRP problem setting from [32].

Over the years, researchers have developed numerous approaches for solving the CVRP, which can be broadly classified as traditional Operations Research (OR) approaches [2, 3, 7–10, 13, 14] and Deep Learning (DL) based methods [21, 24–27, 29, 31, 32, 35]. Conventional OR-based methods typically employ techniques, including exact algorithms, heuristics (classical, metaheuristics algorithms), and hybrid methods, to solve different problem variants. As the name suggests, exact methods aim to evaluate all relevant feasible solutions to determine the optimal tour in the search space. Some commonly used exact algorithms are branch and bound, integer programming, and cutting plane methods[25].

In contrast, heuristic approaches employ carefully-engineered search rules for efficient solution space exploration and thus generate good quality solutions with relatively small computational costs. Metaheuristics are a subclass of heuristic approaches which aim to deeply explore the solution space by exploiting complex neighborhood searches and iterative solution improvements. These methods outperform the former in use cases specifically designed for, as classic heuristics are more generic [19]. All the above traditional OR-based approaches suffer from size and memory limitations as computation complexity increases exponentially for large-scale transportation systems. In real-world scenarios, these methods may not always provide the optimal (or near-optimal) solution in a reasonable amount of time[25].

In recent years, Machine Learning (ML) has emerged as a promising approach to solving real-world problems. Deep-RL is a subset of ML algorithms

that has proven its efficiency in solving VRPs by learning a parameterized model that captures underlying distribution patterns to generate optimal tour solutions. In these approaches, a Deep Neural Network (DNN) based agent is trained to learn an optimal policy, which can be further interpreted as a knowledge-rich search heuristic. The problem is usually set up as either a local search problem with the iterative improvement of an initial solution or a sequential construction process that starts with an empty tour and successively adds customer nodes until a complete solution is obtained [25]. Recent advancements in the domain include the use of complex deep learning architectures, such as attention mechanisms and graph convolutions, to extract more contextual representations of the problem instances[29, 32]. Compared to the OR-based algorithms, the trained policies based on Deep-RL techniques are more efficient in solving large-size problems. However, most of these models are trained on artificially generated datasets that ignore the complex transportation network, in reality, [22]. In this research project, we propose a Deep-RL-based approach for solving a realistic version of the CVRP that aims to overcome contemporary methods' limitations.

1.1 Motivation and Contributions

The recent advancements in computational technologies have enabled researchers in deep learning to learn complex functions that can solve tasks by capturing rich contextual patterns from noisy real-world information[34]. Many researchers have exploited this representation-learning approach to solve combinatorial optimization problems like VRPs. They have proven their efficiency in generating good quality results within reasonable inference times [21, 26]. Furthermore, these models are highly scalable and are learned *end-to-end*, i.e., requiring little or no domain knowledge to construct complex search heuristics. We are deeply inspired by Kool et al. [29], which proposes a transformer style [23] encoder-decoder framework to learn the optimal policy using the REINFORCE algorithm [15]. It is a seminal work in Deep-RL-based approaches for solving the CVRP and can be easily adapted to its other variants. Other inspirations include formulating the problem statement explicitly as a Markov Decision Process (MDP) [32] and the use of graph-based architectures [27, 29, 31] for learning context-rich encodings of input graphs.

Although many of the existing ML-based approaches have exhibited superior performance over a variety of VRP problem settings, most of them exploit some simplifying assumptions regarding the input data distribution, due to which they may fail to model complex real-world scenarios accurately.

For instance, in some works such as [26] and [35], distances between the customer nodes are assumed to be Euclidean, whereas these involve convoluted transportation networks in reality. These methods may therefore have low generalizability over real-world data and hence can be applied to develop enterprise-level solutions only up to a certain extent. On the other hand, in [32], the real distances between the nodes are used, which were accessed via GaoDe map API. Although this method is somewhat realistic, these distances still do not reflect the real-time distances a traveler might take while executing the tour. In complex urban environments, there are a number of factors (such as traffic index, weather conditions, construction work, etc.) that might affect the 'actual' route taken by the agent during the delivery process. Therefore, modeling such dynamic changes in the network features leads to a more realistic implementation of the VRP.

In our research project, we propose a Deep-RL-based methodology to solve the realistic version of VRP by sequential solution construction. The model architecture is inspired by [29] and employs an encoder-decoder type framework to predict the next node, given the agent's current state. Our main contributions can be summarized as follows:

- *Realistic data generation:* We have developed a technique to efficiently generate large-scale realistic datasets that contain delay-adjusted distance features between the nodes. This method allows us to augment the static distance matrices with delay adjustments per the real-time measurements of city traffic level, weather conditions, etc.

- *Edge feature encoding:* We have proposed a modified multi-head attention mechanism to encode the input edge features while computing node-node interactions. Explicitly including the edges helps us to learn richer graph context representations with asymmetric distance matrices.

- *Deep-RL based solution for the CVRP:* We have formulated the problem as an MDP, which leads to better model explainability and ease of code reproducibility and useability. The method was implemented using state-of-the-art Deep-RL libraries (Tianshou, Open AI Gym environment) widely adopted in industry and academia.

- *Performance evaluation:* We evaluate our model by comparing it to relevant advanced DL and OR solvers using a comprehensive experimental assessment protocol that assures fairness in comparison among various methods.

1.2 Problem Setting

As discussed above, various variants of the CVRP may consider constraints on service timings, vehicle capacities, route features, etc., depending upon the complexity of the problem setting. Pick-up and Delivery (PD) problems are a prominent sub-class of CVRPs that find extensive applications in today's e-commerce industry. In addition, some variants of the VRP may involve multiple depots or heterogeneous fleets where vehicles have different capacities or costs associated with them[17]. Figure 1.2 outlines some of the most commonly studied versions of the CVRP and establishes their distinctive correlations with our proposed VRP-DYN approach.

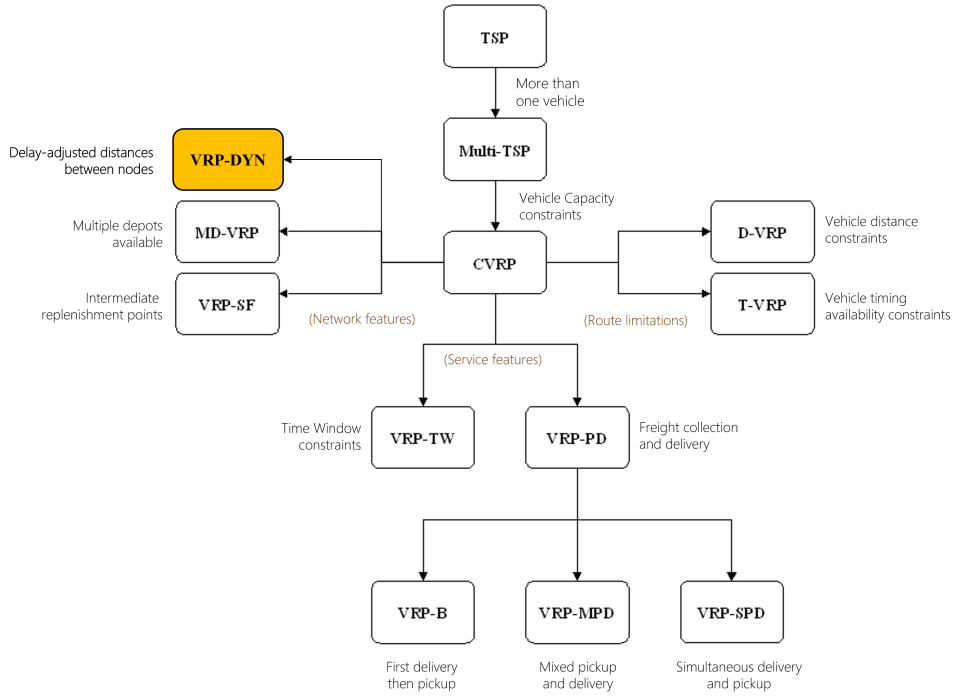


Figure 1.2: VRP-DYN problem setting with other variants of the VRP [17]

Due to the high demand for realistic CVRP solutions in logistics systems, we have formulated our problem setting as an item distribution task. We want to create an efficient transportation management algorithm for our daily grocery supplier. Consequently, the training and evaluation graph instances are sampled from real-life locations of supermarkets, convenience stores, and grocery shops in Berlin. Then, our task is to find the shortest path to fulfill various demands at all customer locations. We assume the vehicle fleet to

be homogeneous, which we think is fair, considering the vehicle fleets are typically owned by the delivery companies.

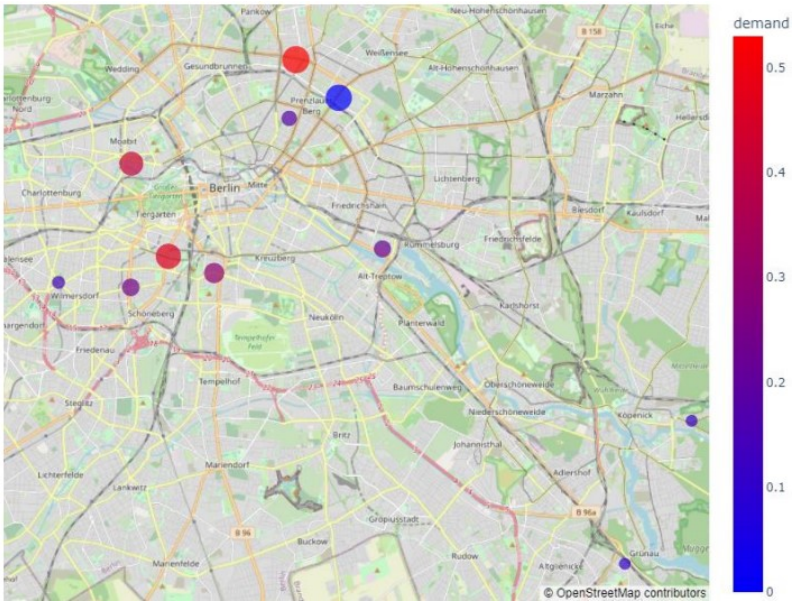


Figure 1.3: Stores in Berlin sampled as customers and respective demands generated stochastically from random Gaussian distribution, visualized as a bubble chart on the Berlin city map.[1]

Furthermore, it is worth noting that the distances retrieved from real-time APIs correspond to the arc distance between the node locations, which contrasts with simplistic Euclidean distance-based approaches. To consider the effect of varying traffic levels, each distance matrix has been augmented with a timing distortion mask - which encodes the adjustment in its transit costs depending on the average traffic congestion in the city. Section 3.3 describes this realistic graph generation process.

1.3 Thesis Outline

So far, we have defined our CVRP setting, our motivation to solve it using an ML-based approach, and our intention to make it more realistic. The remainder of this paper is organized as follows. In Chapter 2, we will review the related literature in combinatorial optimization for both OR-based and ML-based methods. In Chapter 3, we will first present the mathematical

[1]Accessed Overpass API for Open Street Map (OSM) node coordinates and static distance features. https://python-overpy.readthedocs.io/en/latest/introduction.html

formulation of our problem, followed by a detailed description of the model architecture and policy learning procedure. We then define our training procedure and elaborate on the data generation strategy with more realistic graph instances. In Chapter 4, we will explain our experimental process and present the evaluation results. Our findings are discussed in Chapter 5 along with some insights into the future scope. Finally, we will conclude Chapter 6.

Chapter 2

Related Work

More than 60 years have elapsed since Dantzig and Ramser introduced the VRP in 1959 [1]. They described a real-world application concerning gasoline delivery to service stations and proposed the first mathematical programming formulation and algorithmic approach. A few years later, in 1964, Clarke and Wright proposed an effective greedy heuristic that improved the Dantzig–Ramser approach [2]. Following these two seminal papers, hundreds of models and algorithms were proposed for the optimal and approximate solution of the different versions of the VRP. Dozens of software packages for the solution of various real-world VRPs are now available on the market. This interest in the VRP is motivated by its practical relevance and considerable difficulty. In this section, we review some of the OR-based and ML-based methods used to solve the CVRP.

2.1 OR-based Methods

Classical Heuristics

Savings algorithm The Clarke and Wright savings algorithm is a classic OR-based well-known heuristic introduced in [2], which is used to solve the VRPs with both directed and undirected graphs. In this algorithm, the savings are calculated as the first step with $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ for $i, j = 1, ..., n$ and $i \neq j$, where c_{ij} denotes the cost of the route between i and j and s_{ij} denotes the saving of the cost when serving the two routes together instead of serving them separately. Then, there are two main versions when continuing with this algorithm. In the *sequential savings*, one route is repeatedly expanded by adding arcs to the route with respect to decreasing savings until no feasible expansion is available. On the other hand, in *parallel savings*, the

edge with the maximum saving is always merged to any route possible, i.e., all routes are extended in parallel.

The savings algorithm has been widely used in practice and is known for its simplicity and efficiency [30], as well as being a basis for its numerous variants such as parallel savings algorithm with Gaskell [3] and parameterized parallel savings algorithm [7]. Most of these variants targeted lower memory requirements and computation time; however, their improvement was not significant [12]. In [12], it was also shown that the parallel savings algorithm outperforms the sequential version. Other classical heuristics include sequential improvement methods [5, 16], the sweep algorithm [4], and petal algorithms [6, 11]. Since the parallel savings algorithm was shown to outperform the other mentioned heuristics and it is also competent in simplicity and speed [20], we are using the parallel savings algorithm as a baseline for the OR-based methods in the experiments.

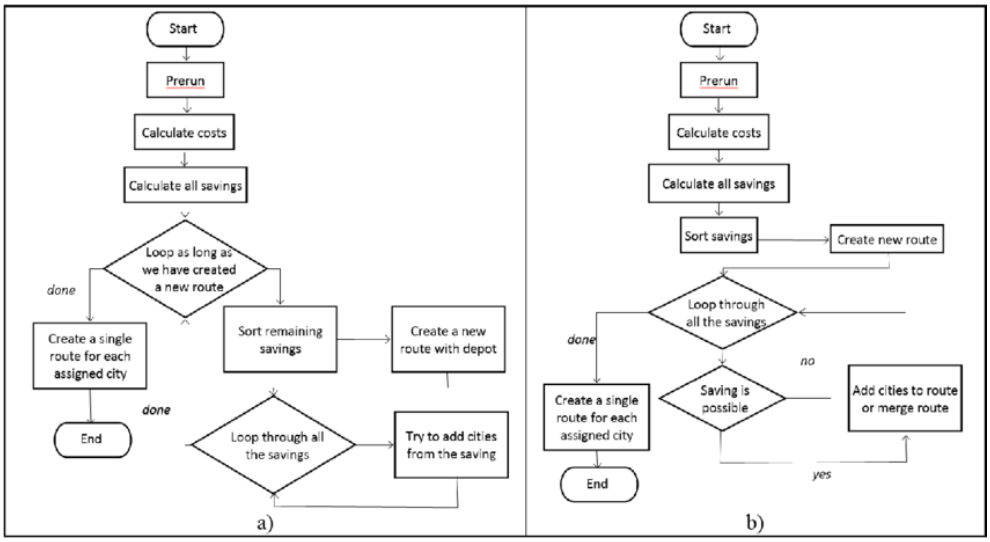


Figure 2.1: The Clarke and Wright savings algorithm[1]. a) sequential version. b) parallel version.

Metaheuristics

Metaheuristics are the procedures where the search space is explored thoroughly by allowing infeasible moves and re-combining solutions, which results in high-quality solutions [20]. There are two main types of metaheuristics: local search methods and population search methods.

[1]Flowcharts for the savings algorithm obtained from [20].

Local search In local search, new solutions are found by moving from the current state to a new state in its neighborhood at each iteration, trying to upgrade the current solution. The most popular local search methods include Tabu Search [18] (TS), Simulated Annealing [10] (SA), and Greedy Randomized Adaptive Search Procedure [9] (GRASP).

Population search In population search, a population of existing solutions are combined together to create better ones and the population is updated with these promising solutions. Genetic Algorithms [13] (GA) and Ant Colony Optimization [14] (ACO) are the well-known examples to the population search methods.

Although metaheuristic methods yield high-quality solutions, they are complex in terms of computation; they take even longer than classical heuristics to find a near-optimal solution. Furthermore, metaheuristics usually depend on the context and require fine parameter-tuning; thus, it can be difficult to implement them in other problem settings [12]. So, ML-based methods have been developed to approximate optimal or near-optimal solutions more efficiently. Also, these methods are mostly used for combinatorial optimization problems with symmetric Euclidean distances, which is not the case in real-world scenarios.

2.2 ML-based Methods

In [24], the Pointer Network (PN) was introduced as the first modern DL model for routing problems, which was trained in a supervised manner to solve the Traveling Salesman Problem (TSP). In their work, a Recurrent Neural Network (RNN) architecture was used for both the encoder and the decoder. The decoder outputs a permutation of the input nodes, and the solution obtained from the OR-Tools[2]was used to supervise the model, which is a limitation of this approach. So, in [21], the authors extended this approach, and a PN was trained to learn a stochastic policy and value functions without supervised solutions, i.e., it was extended to RL. Their approach has given comparable results for small graph sizes ($n = 20$) and even better results for higher graph sizes such as $n = 50$. In [26], the PN was modified to solve the CVRP by replacing the RNN architecture in the encoder with a linear embedding to reduce the computational complexity.

The authors in [31] removed the encoder-decoder architecture to employ a single model based on graph embeddings. Then, in [27], a Graph Neural

[2]Google, OR-Tools. https://developers.google.com/optimization

Network (GNN) was trained in a supervised manner to output a tour as an adjacency matrix. However, it was reported that their model performance is not improved with respect to the previous literature for $n = 20$ [29]. So, the authors in [29] proposed Graph Attention Model to learn a stochastic policy initially developed for the TSP. Nevertheless, the model can extend to other combinatorial optimization problems by changing the input, mask, and decoder context of the new problem instances. In this approach, the encoder has a multi-head attention layer (MHA) based on the Transformer architecture in [23], and the decoder, which is also attention-based, sequentially constructs the tour by choosing the next node to visit at each time step. However, they only input node embeddings in the encoder and assume the distances to be (non-realistic) Euclidean. We will use their model as a baseline in the experiments to compare the effect of including edge features.

The authors in [32] proposed a model similar to [29]. In this work, the encoder uses Graph Convolution Network (GCN), and there are two decoder networks. Moreover, this work uses the shortest geographical distance between two nodes instead of the usual Euclidean distance and shows the importance of the edge features in learning a stochastic policy. They trained the model jointly using RL and supervised learning, where the output of one decoder serves as the label for the other decoder. Their method was evaluated on the real-world datasets and seen to be generalized well [36]; however, there can be a limitation that two models need to be trained.

Most recently, an encoder-decoder network has been developed in [37] to solve the CVRP with node features and edge features, like in [32]. In this work, the Residual Graph Convolutional encoder and Multiple Attention-based decoders (RGCMA) are used. Having multiple decoders is a strategy to construct multiple solutions useful in exploration. The model is then trained by a rollout baseline RL method.

Although these methods can deliver results competitive to OR-Tools in efficient time, they still do not consider the possible real-time variations in the distances, which may reduce the generalizability of the proposed models. Therefore, we aim to generate more robust results in efficient times by solving the CVRP with a more realistic setting using Deep-RL-based methods. In our work, we were inspired by the architecture in [29]. However, we will incorporate edge features and node features to reflect more realistic distances.

Chapter 3

Methodology

3.1 Mathematical Formulation of the CVRP

In the CVRP, all vehicles originate from a single *depot*, which can be denoted as origin O, to fulfill the logistical demands of $N = \{1, 2, \ldots, n\}$ customers. The demand of each customer is a scalar quantity $d_i \geq 0$, which represents the weight of the goods to be delivered at i^{th} node. We have a *homogeneous* fleet of K vehicles to service these customers, each of which has a maximum load-carrying capacity Q. In each generated route, a vehicle starts from depot O, services a subset of customers $S \subseteq N$, and returns back to the depot. The transit cost of moving from node i to j is c_{ij}.

Evidently, this problem can be further structured as a directed graph. Let $G = (V, E)$, where V represents the set of node features for n customers and a depot. E represents the set of asymmetric edges, where each edge $E[i, j]$ denotes the distance between customer i to customer j (i.e. it is the distance matrix). x_{ij}^k is a binary decision variable that equals to one if vehicle k has a path from customer i to customer j, and 0 otherwise. The learning objective for obtaining an optimal solution can thus be stated as:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} \cdot x_{ij}^k \tag{3.1}$$

subject to constraints:

$$\sum_{j \in V \setminus \{i\}} \sum_{k \in K} x_{ij}^k = 1 \quad \forall i \in V \setminus \{0\} \tag{3.2}$$

$$\sum_{j \in V \setminus \{i\}} x_{ji}^k = \sum_{j \in \setminus \{i\}} x_{ij}^k \quad \forall i \in V \setminus \{0\} \quad \forall k \in K \tag{3.3}$$

$$\sum_{i \in V \setminus \{0\}} \sum_{j \in V \setminus \{i\}} d_i \cdot x_{ij}^k \leq Q \qquad \forall k \in K \tag{3.4}$$

$$x_{ij}^k \in \{0, 1\} \quad \forall (i, j) \in A \quad \forall k \in K \tag{3.5}$$

The objective of the CVRP, as defined in (3.1), is to minimize the total distance traveled to fulfill all customers' demands using $k \subseteq K$ vehicles (routes). Constraints (3.2) and (3.3) ensure that each node is visited only once and cannot be revisited by any other vehicle. Constraint (3.4) enforces that the total demand of the visited customers in a route does not exceed the vehicle capacity. The solution to the problem is obtained in the form of multiple *routes* or *tours*, which optimally fulfill the customer demands. A *route* is a sequence of nodes $r_k = [i_0, i_1, i_2, \ldots, i_{r_k}, i_{r_k+1}]$ visited by k^{th} vehicle, where $i_0 = i_{r_k+1} = O$. The total transit cost associated with a route can be represented as $c(r_k) = \sum_{p=0}^{|r_k|} c_{i_p, i_{p+1}}$. The complete solution of a CVRP instance will thus consist of $k \in K$ such feasible routes where each customer has been successfully serviced and visited only once.

3.2 CVRP as a Markov Decision Process

The Markov Decision Process (MDP) is one of the most widely adopted mathematical frameworks for sequential decision-making, which involve stochastic and/or discrete interactions between an environment and an agent[28]. These processes have proven their efficacy in a number of problem settings, ranging from model-predictive control & energy management to combinatorial optimization. As described in the introduction, we have modeled our problem as an MDP that generates solution tours by the sequential construction process. The environment entails the given graph structure for which an optimal policy will be learned - that decides which action to take at the given position such that the total reward (i.e., tour length) is optimized. Specifically, given the state S_t of the environment, the model aims to predict the action A_t that leads to an optimal collection of reward R_t (Fig. 3.1) by the agent.

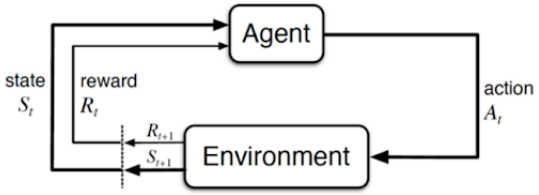


Figure 3.1: A simple graphical representation of the underlying MDP[33]

Since the process is assumed to be Markov, it obeys the Markovian property $p\left(s_{t+1} \mid s_{0:t}, a_{0:t}\right) = p\left(s_{t+1} \mid s_t, a_t\right)$: the next state is dependent only on the current state and action, and is not impacted by the environment states occurring before given time-step[28]. The agent contains a deep reinforcement learning model that parameterizes the next-node probability distribution, which in turn provides the best possible action A_t. The following are the detailed definitions of the five crucial components of our MDP: states space S, action space A, transition rule τ, reward distribution r, and insight into initial state distribution which is defined by the problem set.

State: Let $s_t = (G, V_t) \in S$ be the Markov state of the environment at time t. Since the graph structure for a given instance is fixed, we have dropped the subscript t and define our graph representation as $G = (C, E)$, where $C = \{(c_x^i, c_y^i, d^i, p^i) \mid i = O, 1, 2, \ldots, N\}$ is the set of node-features for N customer nodes plus the depot. For each location on the graph, its node features contain its location co-ordinates (c_x^i, c_y^i), demand d^i and a binary indicator $p^i \in \{0, 1\}$ for depot identification. The adjacency matrix E represents the static distance matrix of the graph G, i.e. the edge $E[i, j]$ corresponds to the arc distance between node i and j. The change in the environment state as per the action taken is tracked with the help of the second component V_t in the state representation s_t. We define the vehicle state $V_t = (m_t, l_t, z_t)$, where m_t denotes the remaining capacity of a vehicle at time t, l_t is the current vehicle position and z_t is an action mask which contains the set of nodes already visited by the agent (partial tour).

Action: In our approach, the action can be interpreted as selecting the next node to visit. We define our discrete action space A, where $a_t \in A$ is the action taken at time step t. Notably, all nodes except the depot can be visited only once, and two consecutive visits to the depot are forbidden. Similar to equations (3.2) and (3.3) in the mathematical formulation, these constraints are enforced in the transition rule described below.

Transition: The transition rule τ defines the change in state s_t of the environment under the action a_t and returns the next state s_{t+1}, i.e. $s_{t+1} = (G, V_{t+1}) = \tau(s_t, a_t) = \tau(G, V_t, a_t)$. As described earlier, the problem graph is static and unaffected by the action. The transitions in vehicle state V_t are governed by the following deterministic rules:

$$\textit{remaining capacity} : m_{t+1} \leftarrow m_t - d_t^i \tag{3.6}$$

$$\textit{current position} : l_{t+1} \leftarrow a_t \tag{3.7}$$

$$action\ mask : z_{t+1} \leftarrow [z_t, index(a_t)] \tag{3.8}$$

Equations (3.6), (3.7) and (3.8) represented the update rule for tracking the remaining capacities of the vehicles and visited nodes during the tour generation. $index(.)$ function returns the index position of the input node and [,] denotes concatenation. Since the transitions are deterministic, $p(s_{t+1} \mid s_t, a_t) = 1$ holds true at each time step t.

Reward: The reward function is the most crucial component of MDP to learn a good policy that converges successfully[28]. Our objective is to generate tour solutions that minimize the total accumulated transit cost; hence we define our rewards to be the negative of the distance between respective nodes. Therefore, the discounted reward function is given as $R = \sum_{t=0}^{T} \gamma^t r_t$[28] where $\gamma \in \{0, 1\}$ is the discount factor and r_t is the current time-step reward, defined as:

$$r_t(s_t, a_t) = -E[l_t, l_{t+1}] \tag{3.9}$$

where a_t action corresponds to the change in agent position from l_t to l_{t+1} and E is the static distance matrix. We want to give equal importance to all the

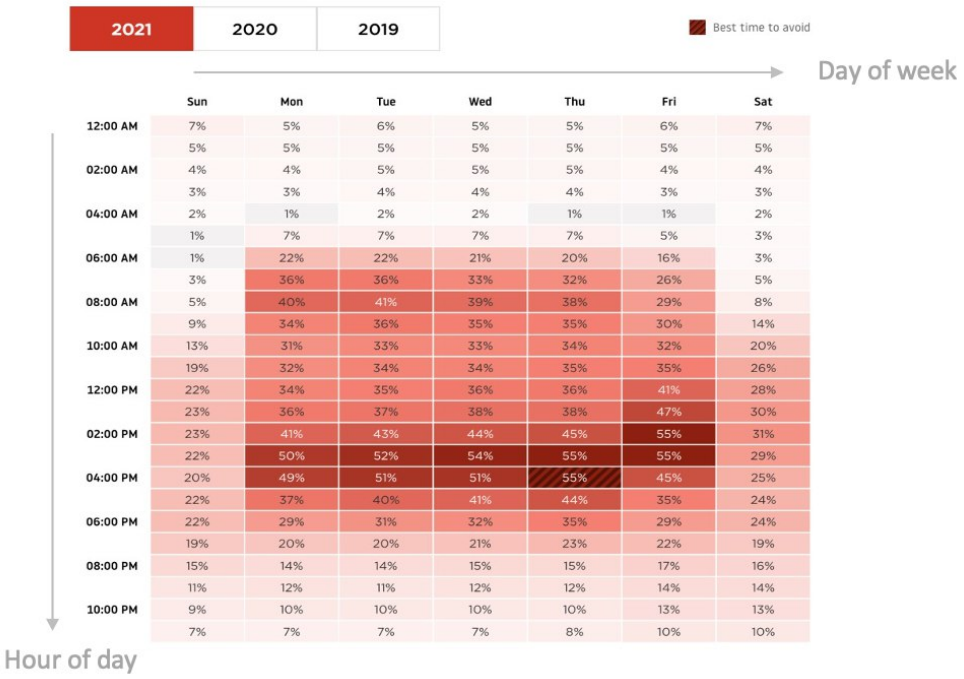
Initial state distribution: Initial distribution is the underlying state distribution from which our problem instances are generated. Each of the independent variables in the input graphs can be thought of as a random variable. Thus, the initial state distribution will be a joint probability density across all such variables. The data generation process is described in detail in the following section, along with some insight into initial distributions.

3.3 Realistic Dataset Generation

In this section, we detail our realistic data generation process that aims to overcome the limitations of existing VRP instance generators by incorporating real-time distances and traffic congestion levels in the city. Specifically, each problem instance is a graph $G = (C, E)$, having a set of node features C and a distance matrix E corresponding to the distribution of customers in the space and their respective demands.

In the operation research literature, several dataset-generation techniques have been proposed for the exact solution of the CVRP. These datasets are created by sampling point coordinates using different probability distributions and conditions [22]. The number of routes is also associated with each

of the datasets. A common distribution used in the literature is the uniform distribution. For instance, in [26], c^i is generated by $c^i_{x,y} \sim U(0,1)$, and demand d^i is generated by $d^i \sim U(1,10)$, where U denotes the uniform distribution. Similarly, in [29], c^i and d^i were generated by $c^i_{x,y} \sim U(a,b)$ and $d^i \sim U(1,10)$ respectively. The problem instances generated by the uniform distribution have the following limitations: the tightness of the instances is high, and instances are too simple and artificial to represent real-world scenarios [22]. Thus, a new set of rules to generate problem instances was proposed by [22]; instead of sampling from a uniform distribution, they sample from a (0,1000) x (0,1000) square grid giving special consideration to the way depot, nodes, and demands are generated. However, most of these employ Euclidean distance metrics for the rewarding process, which can be further interpreted as a symmetric distance matrix with edges derived from node coordinates $E[i,j] = \{(c^i_x - c^j_x)^2 + (c^i_y - c^j_y)^2\}^{1/2}$. While this assumption seems fair for research purposes, the models trained on such unrealistic instances might have low applicability in enterprise-level solutions.



	Sun	Mon	Tue	Wed	Thu	Fri	Sat
12:00 AM	7%	5%	6%	5%	5%	6%	7%
	5%	5%	5%	5%	5%	5%	5%
02:00 AM	4%	4%	5%	5%	5%	4%	4%
	3%	3%	4%	4%	4%	3%	3%
04:00 AM	2%	1%	2%	2%	1%	1%	2%
	1%	7%	7%	7%	7%	5%	3%
06:00 AM	1%	22%	22%	21%	20%	16%	3%
	3%	36%	36%	33%	32%	26%	5%
08:00 AM	5%	40%	41%	39%	38%	29%	8%
	9%	34%	36%	35%	35%	30%	14%
10:00 AM	13%	31%	33%	33%	34%	32%	20%
	19%	32%	34%	34%	35%	35%	26%
12:00 PM	22%	34%	35%	36%	36%	41%	28%
	23%	36%	37%	38%	38%	47%	30%
02:00 PM	23%	41%	43%	44%	45%	55%	31%
	22%	50%	52%	54%	55%	55%	29%
04:00 PM	20%	49%	51%	51%	55%	45%	25%
	22%	37%	40%	41%	44%	35%	24%
06:00 PM	22%	29%	31%	32%	35%	29%	24%
	19%	20%	20%	21%	23%	22%	19%
08:00 PM	15%	14%	14%	15%	15%	17%	16%
	11%	12%	11%	12%	12%	14%	14%
10:00 PM	9%	10%	10%	10%	10%	13%	13%
	7%	7%	7%	7%	8%	10%	10%

Figure 3.2: Average traffic congestion (in %) in Berlin, Germany [1] vs. day of week x hour of the day (2021 data)

[1] Accessed average traffic congestion data via https://www.tomtom.com/traffic-index/berlin-traffic/

We propose a realistic way of generating graph instances for the CVRP. Instead of sampling from $c^i \sim U(a, b)$ as in [26, 29] or from a square grid, we generated prior graph state distribution by querying real-time GIS coordinates (latitude, longitude) of all the grocery stores in Berlin city in Germany - accessed from the Overpass API for Open Street Map (OSM). It is clearly evident that the method can be easily adapted to include any locations of interest or a custom set of coordinates. A query of all grocery & convenience stores in Berlin city returns over 1,800 coordinates, which is the distribution from which our problem instances are generated. To create a graph instance, we randomly sample $N + 1$ node coordinates from our initial distribution and assign one of them as the depot. The customer demand is generated by $d^i \sim U(0, 0.5)$, and we obtain the distance matrix E from Routingpy[2], which returns the real distances and travel time between the nodes of our graph G.

Delay-adjusted edge features: The travel times obtained from OSM are static, i.e., correspond to the transit times at the querying time. We aim to make the edge features more realistic by adjusting their values in order to capture complex and dynamic transportation scenarios. To this extent, we propose a novel approach to augment the graph instances with adjusted distance features per the real-time information about the traffic level in the city. The edge matrix E can be considered a linear combination of two matrices $E = D \odot \Delta D$, where $\odot$ represents the element-wise matrix product. Here, D is the static distance matrix obtained from the Routingpy API as described above, and ΔD is defined as the 'distortion' matrix that contains adjustment factors for the corresponding distance features (which depend upon traffic congestion level). Given that the average traffic congestion in the region is μ_τ, the distortion matrix can be constructed stochastically as:

$$\Delta D[i, j] \sim \mathcal{N}(\mu = \mu_\tau,\ \sigma^2 = 0.2\mu_\tau) \tag{3.10}$$

which codifies the underlying assumption that the traffic congestion level impacts the distance features as per a normal distribution (in spatial dimensions). In this way, the final edge matrix E contains distance values adjusted as per the traffic. It is worth noting that this technique can be further extended to include more real-world information (e.g. weather data) in the network features. Fig. 3.2 tabulates the traffic congestion levels in Berlin for 24 hours x 7 days. Finally, for creating our dynamic edge feature dataset, we augment each of the static graphs with 24 distortion masks ΔD_τ which correspond to 24 hours of the day (modeling that the solution tour is to be executed at that hour).

[2]https://pypi.org/project/routingpy/

3.4 Deep RL-based Solution for VRP-DYN

In this section, we describe our deep reinforcement learning (DRL) based approach for solving the VRP-DYN problem setting by explicitly encoding the dynamic edge features. Fig. 3.3 depicts an outline of the proposed framework, which employs an attention-based encoder-decoder architecture to learn the optimal policy. The solution tour $\pi = (\pi_1, \pi_2, \ldots, \pi_T)$ is generated by sequential decoding process, where $\pi_t \in \{O, 1, 2, \ldots, N\}$ such that each customer is visited once and depot O may occur multiple times. The policy network models a stochastic policy $p(\pi \mid s)$ for action decision making, which is parameterized as:

$$p_\theta(\boldsymbol{\pi} \mid s) = \prod_{t=1}^{n} p_{\boldsymbol{\theta}}\left(\pi_t \mid s, \boldsymbol{\pi}_{1:t-1}\right) \qquad [29] \qquad (3.11)$$

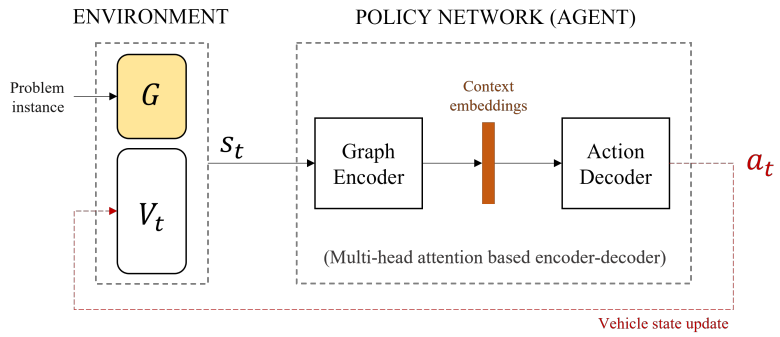


Figure 3.3: Outline of DRL-based solution of VRP-DYN problem. The policy network contains an encoder-decoder style model $p_\theta(\boldsymbol{\pi} \mid s)$, which first extracts the graph context features as per the input state s_t and then selects the best possible action a_t through an autoregressive decoding process.

The policy network thus returns the next node probability distribution, based upon which the action a_t can be chosen as per the maximum a posteriori (MAP) value or can be sampled directly from $p_\theta(\pi|s)$. We have adopted the former, i.e. select the highest probability node as the next node to visit. As explained in the introduction, we are highly inspired by the use of Transformer style encoder-decoder architectures that have widely proven their efficiency in various domains such as computer vision, recommendation systems, and combinatorial optimization problems. Our model framework is derived from [29, 32] which can be interpreted as a Graph Attention Network that takes graph states as input and generates the optimal solution tour autoregressively. The encoder consists of a stack of multi-head attention (MHA)

layers [23] that encodes the graph state s_t and produces contextualized graph embeddings. This attention mechanism is a crucial component of the architecture and aims to learn the graph context features which capture complex node-node interactions from its state. The decoder takes these graph representations as input and sequentially outputs the customer to be visited at each time step, i.e. the action a_t. The vehicle state V_t keeps track of the partial tour $\pi_{1:t-1}$ such that the next node probabilities are masked during the decoding process - as per the nodes which have already been visited. The following sections describe the network architecture and training protocol in detail.

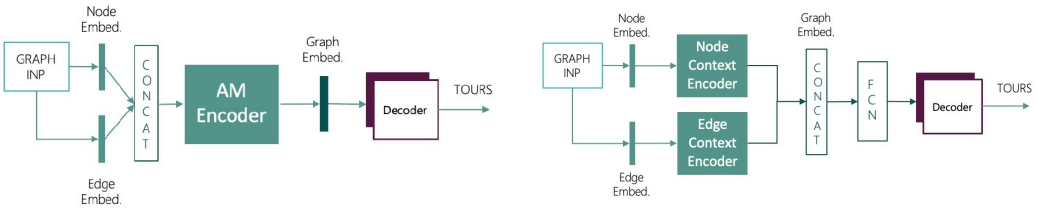
Model Architecture: Encoder

The encoder takes raw features of the problem instance graph (depot/customer locations, demands, etc.) as input and computes contextualized graph embeddings of fixed dimensionality d_h. It consists of a sequence of N attention modules (MHA) [23], each consisting of a graph attention mechanism followed by batch normalization and a feed-forward layer. The raw graph features x^i are first converted to initial node embeddings by a linear projection $h_i^(0) = W^x x^i + b^x$, where $h_i^{(0)}$ represents the initial input embedding for i^{th} node. Each of the attention modules updates the node embeddings (as in a GCN) by computing node-node interactions as a combination of node and edge feature attentions. The final graph context embeddings are thus obtained as the mean of node embeddings after propagating through N attention modules: $\overline{\mathbf{h}}^{(N)} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{h}_i^{(N)}$ where $\overline{h}^{(N)}$ is the learned graph context, which is fed to the decoder along with all the embeddings $h_i^{(N)}$.

Basic attention module (node feature encoding): The architecture for each of the attention modules is inspired from [23, 29] where each layer consists of the following sublayers: a multi-head attention (MHA) layer that computes interactions between the nodes followed by batch normalization for more efficient training. The MHA layer is followed by a feed-forward layer and contains skip connections to propagate low-level activations. This module is similar to a graph attention layer which updates node features as per the weighted network interactions. Let $h^{(l)}$ denote the node embedding computed at layer $l \in \{1, 2, \ldots, N\}$, which can be derived as follows:

$$\begin{aligned}
\hat{\mathbf{h}}_i &= \mathrm{BN}^{\ell}\left(\mathbf{h}_i^{(\ell-1)} + \mathrm{MHA}_i^{\ell}\left(\mathbf{h}_1^{(\ell-1)}, \ldots, \mathbf{h}_n^{(\ell-1)}\right)\right) \\
\mathbf{h}_i^{(\ell)} &= \mathrm{BN}^{\ell}\left(\hat{\mathbf{h}}_i + \mathrm{FF}^{\ell}\left(\hat{\mathbf{h}}_i\right)\right)
\end{aligned} \qquad \text{[29]} \qquad (3.12)$$

While the above encoder architecture enables us to propagate the node-wise information, it does not include the edges (or distance features) during the feature extraction process. These real-time distances, contained in the edge feature matrix E, can serve an important role in action decision-making as they directly impact which node can be visited next. Furthermore, explicitly including these (dynamic) edges in the attention computation overcomes the limitations of existing approaches that exploit Euclidean distance methods for node feature propagation. The design of this module is described in the following section.



(a) TM01: First combine the initial node and edge embeddings and then encode for generating graph context.

(b) TM02: Independently encode the node and edge features and then combine them for creating graph context.

Figure 3.4: Trial configurations for edge feature encoding

Modified attention module (node+edge feature encoding): Each of the multi-head attention layer $\text{MHA}_i^{\ell}(h_{1:N}^{\ell-1})$ in the existing formulation takes only the current node embeddings $h_{1:N}^{\ell-1}$ as input, and thus excludes the edge features while attention computation. We propose a modified version of this attention mechanism which also takes the edge features as an input to the MHA layer. It was a challenging task for us to explicitly encode the edge compatibility during the feature extraction. Fig. 3.4 depicts such trial configurations that were implemented during our research. However, both of these approaches were found to be ineffective during initial experimentation and hence were discarded as a final solution for edge feature encoding. From these learnings, we came to know that an effective way to combine edge features with node interactions will be to directly include them in the compatibility computation. Therefore, we introduce the following multi-head graph attention (MHGA) layer which incorporates both node and edge features (delay-adjusted distances) during attention computation:

$$Q = h^{(l)}W_l^Q, K = h^{(l)}W_l^K, V = h^{(l)}W_l^V \tag{3.13}$$

$$A_{edge} = \text{LinTransform}(E) \tag{3.14}$$

$$\text{MHGA}_i^{\ell}(h_{1:N}^{\ell-1}, E) = \text{softmax}\left(\frac{QK^T + A_{edge}}{\sqrt{d}}\right) \tag{3.15}$$

where A_{edge} can be interpreted as an adjacency matrix that contains edge compatibilities, which are obtained by a linear projection LinTransform(.) of the edge features. This technique allows us to encode the edge features and thus learn a combined graph context representation to be used for the decoding process (refer to Fig. 3.5). The rest of the layer architecture is the same as described in the previous section. Thus, the real-time distance information has been included not only in the training process but also during the feature extraction process in the encoder network.

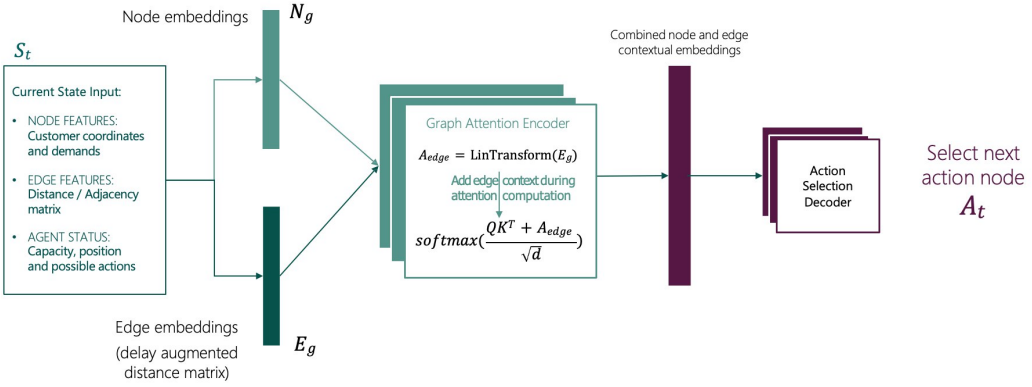


Figure 3.5: Encoder-Decoder Network for solving the CVRP problem

Model Architecture: Decoder

The decoding process is sequential, i.e. it predicts the best action to take at every time step $t \in \{1, 2, \ldots, T\}$ as per the policy $a_t \sim \pi_\theta(a_t \mid s_t)$. The decoder architecture is adapted from [29], which first represents the graph context in the form of a context node c and then computes the next-node log probabilities based on the attention between graph context and node embeddings. The vehicle state vector V_t in the environment is used to mask out the nodes that have already been visited during the partial tour $\pi_{1:t-1}$. Fig. 3.6 illustrates the decoding process in detail: at each time step t, the decoder selects the next node with a combination of graph embeddings and vehicle state as the context node.

Graph context embedding: The graph context at each time step t is obtained as the combination of encoded graph embeddings and node embed-

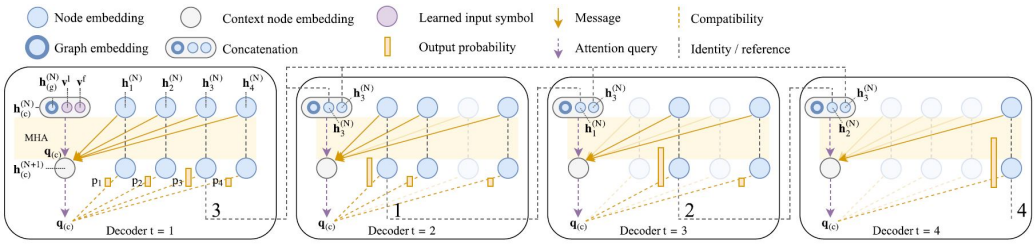


Figure 3.6: Illustration of the sequential decoding process for a sample tour $\pi = (3, 1, 2, 4)$ [29]. The input to the decoder is aggregated graph embeddings and all the node embeddings after N-layer propagation in the encoder.

dings of the first and the last visited node. As described in section 3.2, the vehicle state representation V_t contains the current agent position l_t at each time. The graph context is thus given as:

$$\mathbf{h}_{(c)}^{(N)} = \begin{cases} \left[\overline{\mathbf{h}}^{(N)}, \mathbf{h}_{l_t}^{(N)}, \mathbf{h}_{\pi_1}^{(N)}\right] & t > 1 \\ \left[\overline{\mathbf{h}}^{(N)}, \mathbf{v}^1, \mathbf{v}^{\mathrm{f}}\right] & t = 1 \end{cases} \qquad [29] \qquad (3.16)$$

where $[\cdot, \cdot, \cdot]$ represents the horizontal concatenation of various embeddings, each of which is a d_h-dimensional vector. The graph context $\mathbf{h}_{(c)}^{(N)}$ can be thought of as a hypothetical context node that encodes the information about the underlying graph structure as well as the current vehicle state. The dimensionality of the graph context is brought back to d_h during the attention computation. The context is first propagated with the help of a single MHA layer, followed by the calculation of log probabilities as follows.

From graph context to log-probabilities: Given the node context $h_{(c)}$, we first compute its multi-head attention compatibility with all the nodes, with a score of $-\infty$ for the nodes that have already been visited. Mathematically, the compatibility score can be computed as

$$u_{(c)j} = \begin{cases} \frac{\mathbf{q}_{(c)}^T \mathbf{k}_j}{\sqrt{d_{\mathrm{k}}}} & \text{if } j \neq \pi_{t'} \quad \forall t' < t \\ -\infty & \text{otherwise} \end{cases} \qquad [29] \qquad (3.17)$$

where $q_{(c)} = W^Q \mathbf{h}_{(c)}$ is the context node query and $\mathbf{k}_i = W^K \mathbf{h}_i, \quad \mathbf{v}_i = W^V \mathbf{h}_i$ are keys and values obtained from encoded node embeddings. The dimensionality of $d_k = 8$ for $d_h = 64$ and 8-head attention mechanism. It is worth noting that batch normalization and skip connections have not been used during decoding, similar to the *glimpse* method in [21]. The final outputs (next-node probabilities) are then computed with a single-head atten-

tion layer with the tanh clipping method. The computation is similar to Eq. (3.17), except that the results are clipped within interval $[-C, C]$ before masking.

$$u_{(c)j} = \begin{cases} C \cdot \tanh\left(\frac{\mathbf{q}_{(c)}^T \mathbf{k}_j}{\sqrt{d_\mathrm{k}}}\right) & \text{if } j \neq \pi_{t'} \quad \forall t' < t \\ -\infty & \text{otherwise.} \end{cases} \qquad [29] \qquad (3.18)$$

where $u_{(c)j}$ is the final unnormalized probability score for the next node selection (logit vector). The final output of the model can thus be computed after a softmax transformation of the compatibility scores as:

$$p_i = p_{\boldsymbol{\theta}}\left(\pi_t = i \mid s, \boldsymbol{\pi}_{1:t-1}\right) = \frac{e^{u_{(c)i}}}{\sum_j e^{u_{(c)j}}} \qquad [29] \qquad (3.19)$$

Training Algorithm: the policy gradient approach

The model presented above has a framework for generating the optimal policy with reinforcement learning. We have used the Monte Carlo Policy Gradient (REINFORCE) algorithm with a greedy rollout baseline proposed by [29] to train this model. We define the policy objective function as follows:

$$\nabla_\theta L(\theta) = \mathbb{E}_{\sim \rho_\theta(\pi|s)}\left[\nabla_\theta \log \rho_\theta(a|s)(L_\theta(\pi)] - b(s)\right) \quad [29] \qquad (3.20)$$

In the REINFORCE algorithm with a greedy rollout baseline, there are two policy networks: In the first policy $p(\pi|s)$ we sample actions from our logits; in the second policy, called baseline $b(s)$ [29], we always act greedily. Because when we sample action from the logits from the base model over several steps, the training becomes unstable and noisy; hence, we need a baseline model which is independent of action to reduce the variance during training. The weights of the base model are copied into the baseline model after n steps of training is completed[28]

Algorithm 1: REINFORCE with Greedy Rollout Baseline [29]

Input: Number of epochs E, steps per epoch T, batch size B, significance α
Initialize policy parameters θ and baseline policy parameters $\theta^{BL} \leftarrow \theta$
for epoch $= 1, ..., E$ **do**
  for step $= 1, ..., T$ **do**
    $s_i \leftarrow \text{RandomInstance}() \ \forall i \in \{1, ..., B\}$
    $\pi_i \leftarrow \text{SampleRollout}(s_i, p_\theta) \ \forall i \in \{1, ..., B\}$
    $\pi_i^{BL} \leftarrow \text{GreedyRollout}(s_i, p_{\theta^{BL}}) \ \forall i \in \{1, ..., B\}$
    $\nabla \mathcal{L} \leftarrow \sum_{i=1}^{B} (L(\pi_i) - L(\pi_i^{BL})) \nabla_\theta \log p_\theta(\pi_i)$
    $\theta \leftarrow \text{Adam}(\theta, \nabla \mathcal{L})$
  end
  if $\text{OneSidedPairedTTest}(p_\theta, p_{\theta^{BL}}) < \alpha$ **then**
    $\theta^{BL} \leftarrow \theta$
  end
end

Chapter 4

Experiments

4.1 Experimental Setup

Dataset Generation

As discussed in Section 3.3, our datasets consist of a set of problem instances, each being a graph $G = (C, E)$ with node feature matrix C (having latitude, longitude, and customer demand information) and edge feature matrix E (containing distances between nodes). These were defined as (N, d_C) and (N, N) dimensional matrices (one for each problem graph), where N is the number of nodes and d_C is the dimensionality of node features. In our case, $d_C = 4$ as each node has 4 features associated with it: 2 coordinates (c_x^i, c_y^i), demand d^i and a binary indicator p^i which is 1 for depot and 0 for customers. For a real-world logistical problem, we queried the locations of grocery stores, convenience stores, etc. in Berlin from Overpass API[1], which resulted in a global node pool of $\tilde{1}$,800 coordinates. We then generated problem instances of size N by randomly sampling from this prior distribution. One of the nodes is randomly assigned as the depot and customer demands are sampled from the distribution $d^i \sim U(0, 0.5)$. For a total (normalized) vehicle capacity $Q = 1$, the customer demands are assumed to be $d^i \geq 0.5Q$ which we think is a fair assumption for realistic logistical systems. Node coordinate features are also normalized in the $(0, 1)$ range before feeding into the model encoder.

In addition, for obtaining instance-wise distance matrices E, we first generated a global distance matrix of size (1800, 1800) and then retrieved per instance distances by a custom subsampling procedure. This technique has allowed us to reduce retrieval redundancy and is faster than querying the

[1] https://python-overpy.readthedocs.io/en/latest/introduction.html

distance matrices iteratively. Routingpy[2] library enabled us to access the real-time distance and transit time values between node-node pairs for a given set of problem coordinates. Another aspect of the realistic data generation process included assigning a delay-adjustment mask ΔD_τ to each graph instance. As discussed in Section 3.3, we will create 24 replicas of each base graph, corresponding to the distance delay adjustments for 24 hours of a day. Given the average traffic level in the city at the given hour is μ_τ, the distortion matrix is derived from a normal distribution $\Delta D[i,j] \sim \mathcal{N}(\mu_\tau, 0.2\mu_\tau)$. For instance, if the trip has to be made at 8:00 AM on a Saturday, and the traffic level in the city is $\mu_\tau = 10\%$, then distance adjustments are sampled from $\mathcal{N}(0.1, 0.02)$, i.e., the delay adjustments normally lie (across node space) in the interval $(0.08, 0.12)$. This process is repeated for 24 different hours of the day to augment each of the graph instances with delay adjustment matrices.

We generated 10,000 problem instances of size S10, S20, S30, and S50 where S represent static edge features. i.e., they are the raw distance features extracted from OSM. For delay-adjusted edge feature datasets, we created 24 replicas of each of the 10,000 problem instances per the sampling process described above. These sets of data are referred to as D10, D20, D30, and D50, where D represent *dynamic*, i.e., they are the distances that are taken from the OSM, which are then distorted to create 24 variants for every hour of the day to account for the real-time deviations in the distances. Each of the above datasets has a corresponding test set with 10,000 instances of the same graph size.

Implementation Details

Our implementation was done in the Tianshou framework, a Deep RL library that enables us to use vectorized environments for flexible implementations of the policy learning process. The graph environments were implemented as OpenAI Gym (now Gymnasium), a standard development library for DRL and game-playing agents. The use of Tianshou was beneficial for efficient sampling, multiple experience collection, episode tracking, and parallelized code execution. We have used the following configurations for all our experiments: 500 environments (vectorized); epochs trained = 1,000, with a batch size of 32 and a buffer size of 50,000. For training the model, we use the Adam optimizer after hyperparameter tuning on S10 datasets. The learning rate was set to $2.2 \cdot 10^{-5}$ after a stratified model tuning process with Raytune[3]. In our policy network, we used 64 embedding dimensions and 16

[2]https://pypi.org/project/routingpy

[3]https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html

hidden dimensions in our multi-head attention. The encoder contains $N = 3$ layers, which seems to be a good trade-off between computational & model complexity and was found to be an appropriate value after the parameter-tuning process. The exact model configurations are provided in Appendix B. The model was trained on a 30GB RAM cloud machine with 8 CPUs and P5000 GPU with 16GB RAM.

Baselines

We compare our model with the OR-based methods, the Parallel Savings algorithm is selected as a baseline. The most relevant baseline model is the Attention Model (AM) of [29], which employs a similar attention mechanism with node feature encoding. We have re-implemented the AM model in our training environment to maintain fair comparability of the results. The AM model was fine-tuned using a grid search for hyperparameter tuning. In both AM and our Graph Model (GM), the training dataset is split into batches of size 32, and the models are trained for 1,000 epochs.

Evaluation Metric

In the evaluation process, the mean tour length aggregated along all the instances is used as the comparison criterion. In comparing model performances, a lower value of the average mean tour length is better, corresponding to lower transit costs and better profit margins. We will also track the gap (%age difference vs. baseline) compared to the ML-based and traditional OR approach.

4.2 Experimental Results

We compare the performance of our proposed Graph Model (GM) vs. the Attention Model (AM) proposed in [29] on the realistically generated *static* and *dynamic* problem sets. For assessing the effect of graph size, we arrange our results in increasing N order. Table 4.1 shows the aggregated mean of test tour lengths for various solution implementations, along with the gap (%age difference). The results can be compared along the columns - the best value for each problem size is bold and underlined, and the second best is underlined. The gap value is calculated as $gap = (L_{Ours} - L_{Base})/L_{Base}$.

These results have been obtained from test tours generated by the model trained for 1,000 epochs on *dynamic* distance graphs, which is our primary problem setting. Our proposed model performs significantly better than

Model	**D10**	**D20**	**D30**	**D50**
Ours (GM)	**105.13**	**219.05**	**321.95**	522.22
Baseline (AM)	108.44	226.95	325.70	**515.72**
Parallel Savings	140.86	253.36	360.72	641.84
Gap_{ML}	-25.37%	-13.56%	-10.74%	-18.37%
Gap_{OR}	-3.06%	-3.51%	-1.15%	0.48%

Table 4.1: Comparison of average test tour length results (in km) for proposed model vs. baselines (*dynamic* instances with $N = 10, 20, 30, 50$)

the AM baseline, with a gap of about 5% in most cases. The gap seems to increase marginally over the increasing graph size. Another important metric to consider may be the average tour length per customer node, i.e., on average, how much a vehicle will be traveling to serve a customer - as per all the test solutions generated by the corresponding method. As per the tour solutions generated by our proposed method, a vehicle has to travel 12.29 km to deliver a grocery shipment to one of its customers (in Berlin city). This transit cost is about 4.4% higher for the tour solutions generated by the AM baseline method (= 12.84 km/customer).

Compared to the OR-based solver (Parallel Savings, Clarke & Wright), our DRL method performs notably better for small graph sizes ($N = 10, 20$). Our model performance is slightly lower than the OR solver for larger graph sizes - as the gap is marginally positive. This can be attributed to our proposed graph attention module being relatively less scalable than the OR methods. The computational complexity of the policy networks also limits the scalability of ML approaches. One important point is that the current GM model has been trained for 1,000 epochs and might have performed even better if trained for convergence with larger training dataset sizes (50k or 100k, for instance). However, such implementations required larger resource availability and were not possible for the limited scope of this project as a proof-of-concept of our proposed approach.

Table 4.2 shows the same evaluation metrics for *static* distance graphs calculated on the test datasets. In this problem setting, our model performance is also comparable to the AM baseline method. However, the gap %

as per the problem size is significantly lower than that of the experiments with the dynamic dataset. This establishes that our proposed method is more efficient in solving realistic graph instances containing delay-adjusted edge features. Similar to the previous observations, OR solver is marginally performing better than the GM algorithm. The transit cost per customer is 10.86 km for our solution and 10.55 km for AM baseline test tours.

Model	**S10**	**S20**	**S30**	**S50**
Ours (GM)	**102.56**	**172.62**	**253.31**	**455.57**
Baseline (AM)	105.47	180.95	286.52	461.17
Parallel Savings	115.96	210.09	298.19	535.26
Gap_{OR}	-11.55%	-17.83%	-15.05%	-14.86%
Gap_{ML}	-2.76%	-4.61%	-11.59%	-1.214%

Table 4.2: Comparison of average test tour length results (in km) for proposed model vs. baselines. (*static* instances with $N = 10, 20, 30, 50$)

It is worth noting that the average transit cost per customer was about 13.2% higher for the *dynamic* dataset solutions generated by our method. However, the difference between the per-customer transit costs for *static* vs. *dynamic* dataset is significantly higher for the AM baseline method (21.8%). This shows that our approach can successfully capture the dynamic correlations in the delay-adjusted edge features, leading to better solutions with a lower logistical cost per customer. Whereas the baseline approach fails to cope with the dynamic changes in traffic levels, leading to increased transit cost per customer served.

The epoch-wise training curve for both GM and AM models ($N = 10$) is shown in figure 4.1, which depicts that the training curve approaches convergence. The loss values almost stabilize for the AM baseline after about 700 epochs. However, the training loss for our model seems to decrease until max_epoch is reached.

Figure 4.2 depicts the epoch-wise moving average of test rewards for both GM and AM models ($N = 10$). For the AM model, the average reward value stagnated after some training for a while, whereas that continued slightly increasing until the end for the GM model. This observation is in conjunction

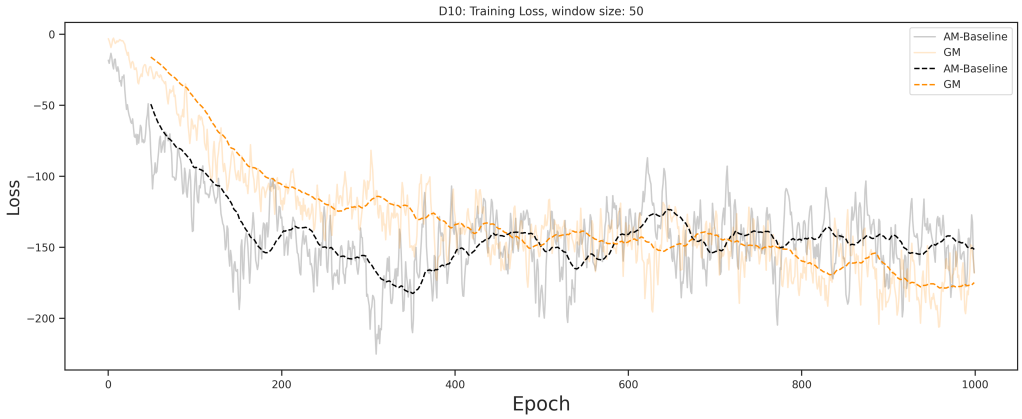


Figure 4.1: The training loss curve on dynamic distance graphs of size 10.

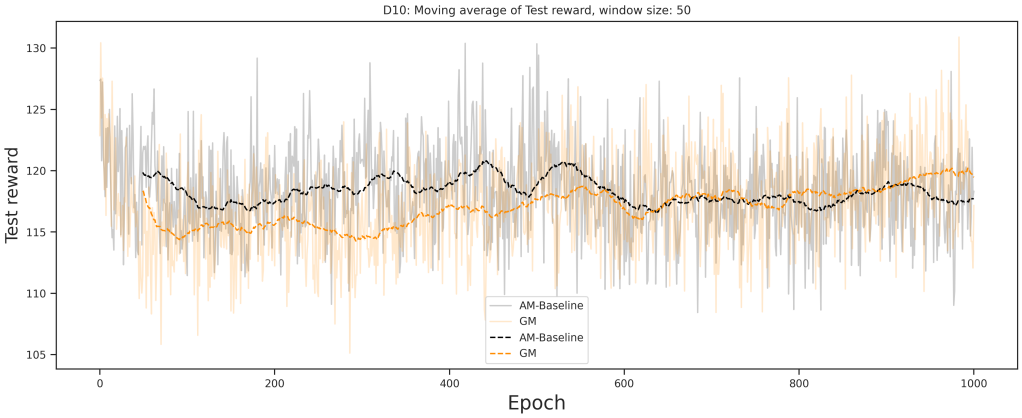
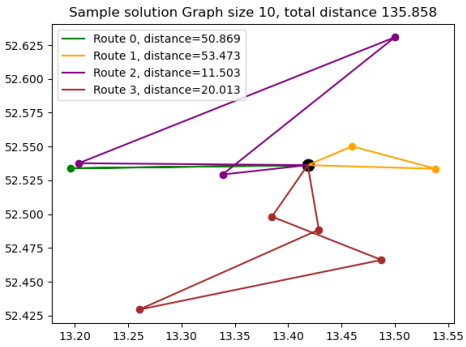


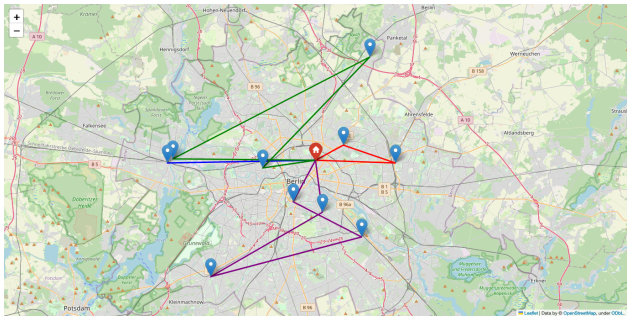
Figure 4.2: The average test reward on dynamic distance graphs of size 10.

with the corresponding training curves. The Appendix includes all such training curves for other problem settings.

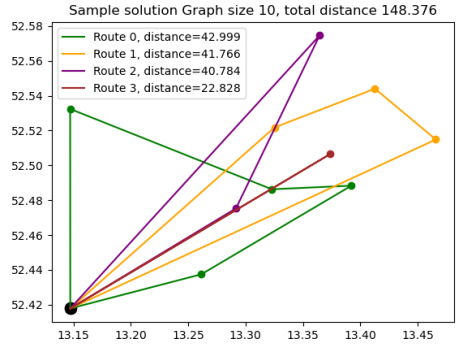
4.3 Sample Tour Visualizations



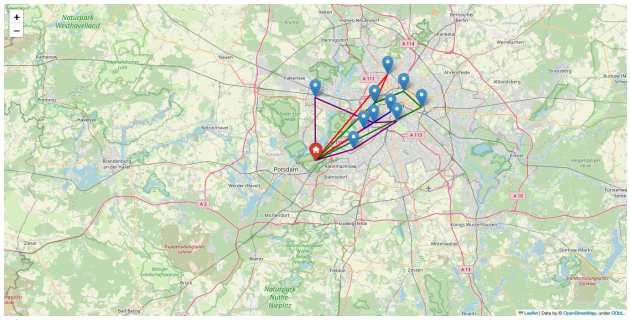
(a)



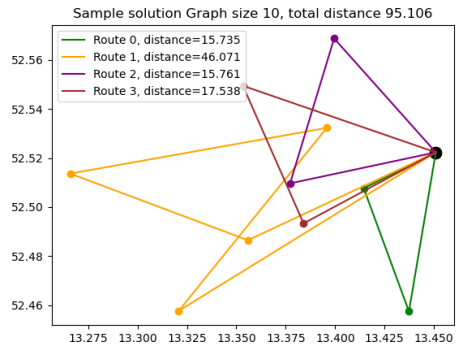
(b)



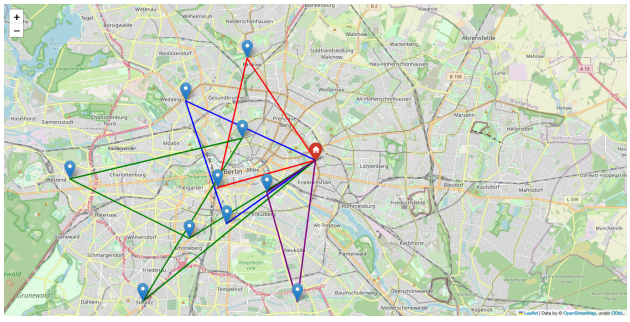
(c)



(d)



(e)



(f)

Figure 4.3: Sample solutions generated greedily, for graph size of 10 using our novel generated data. Left: Visualization using matplotlib. Right: Route visualization on the map.

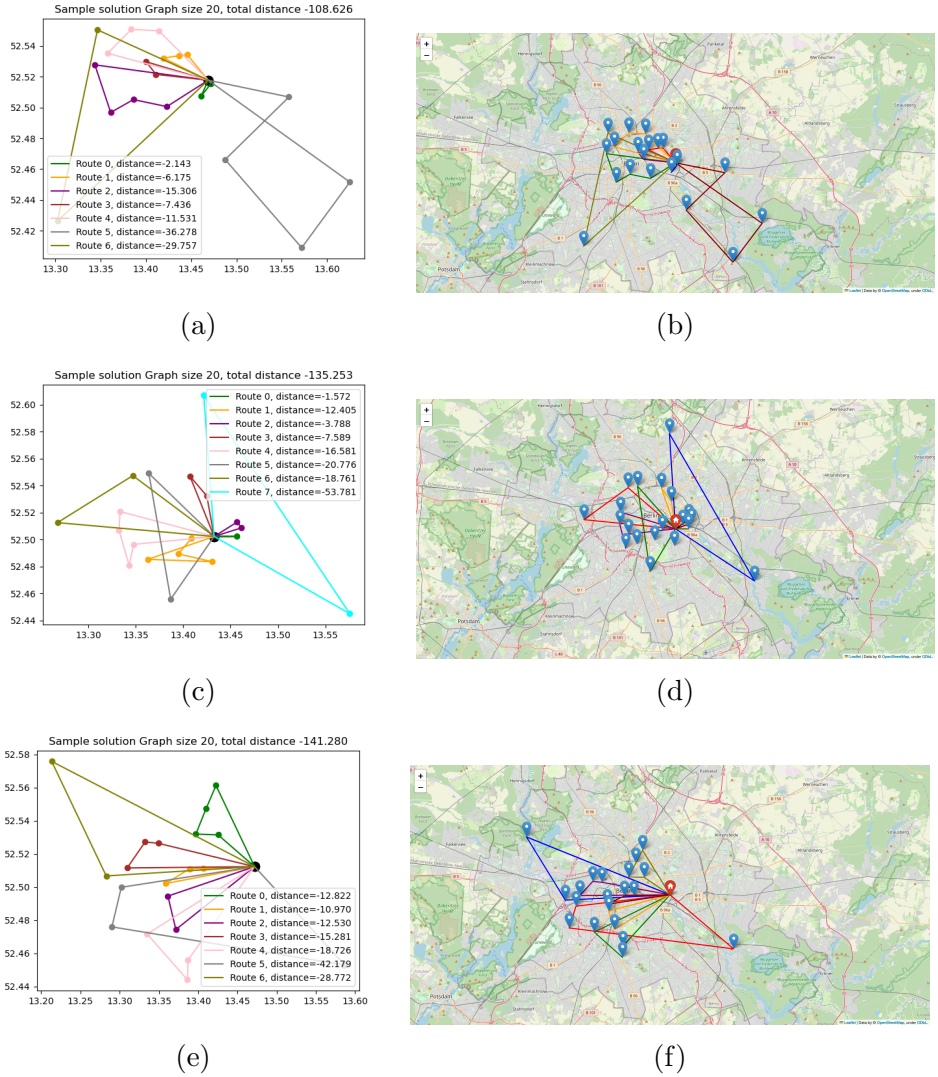
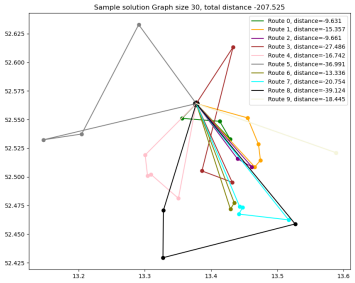
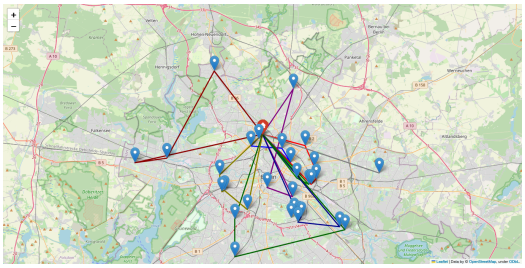


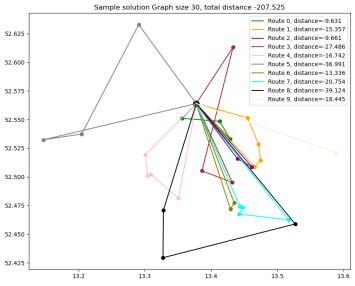
Figure 4.4: Sample solutions generated greedily, for graph size of 20 using our novel generated data. Left: Visualization using matplotlib. Right: Route visualization on the map.



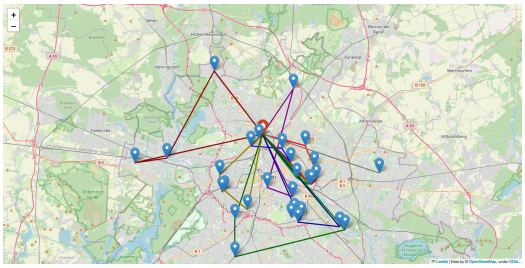
(a)



(b)

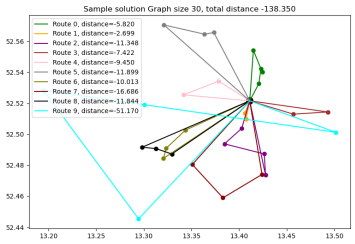


(c)

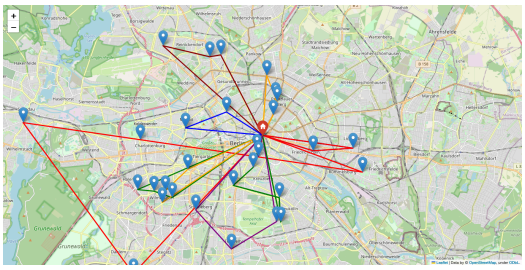


(d)

Figure 4.5: Sample solutions generated greedily, for graph size of 30 using our novel generated data. Left: Visualization using matplotlib. Right: Route visualization on the map.

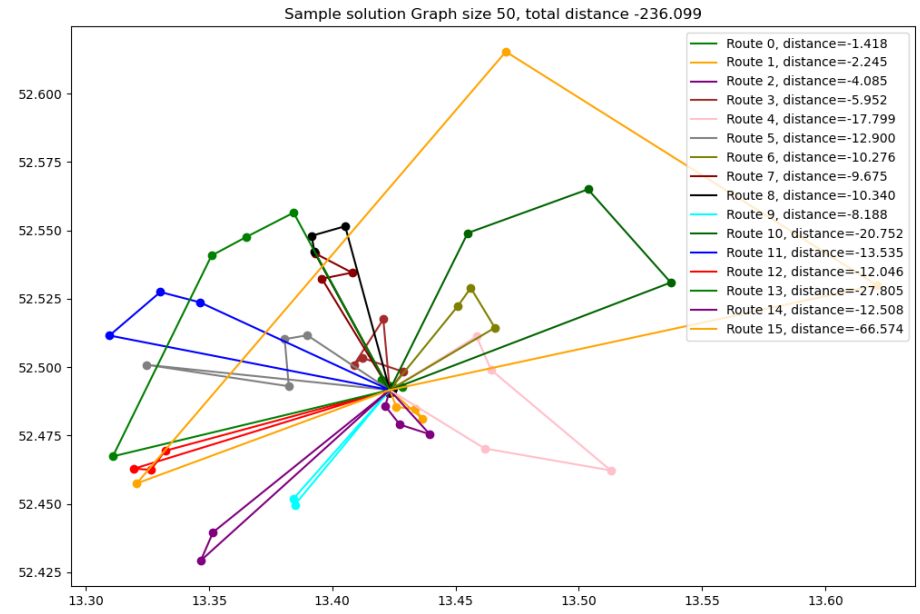


(a)

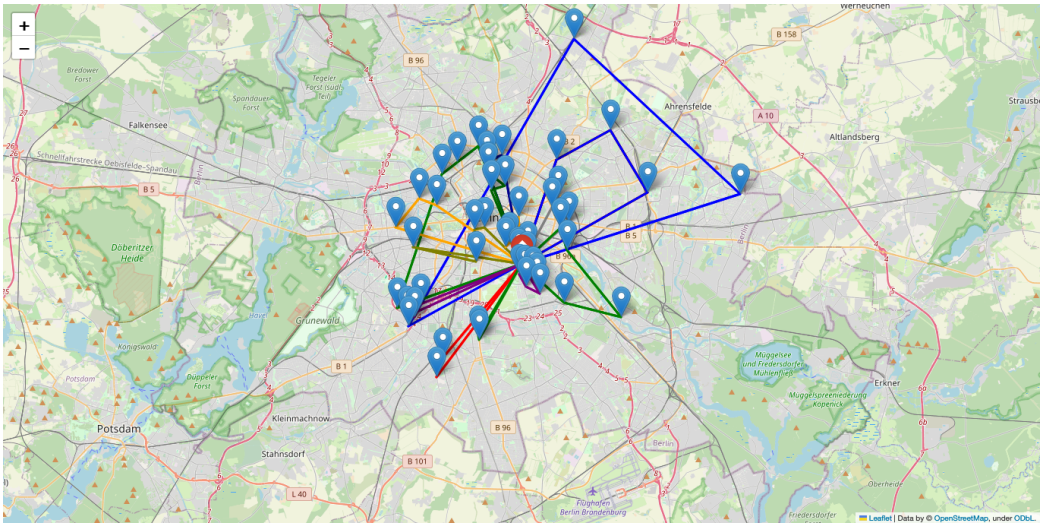


(b)

Figure 4.6: Sample solution generated greedily, for graph size of 30 using our novel generated data with delay-adjusted edge features. a) Visualization using matplotlib, b) Route visualization on the map.



(a)



(b)

Figure 4.7: Sample solution generated greedily, for graph size of 50 using our novel generated data with delay-adjusted edge features. a) Visualization using matplotlib. b) Route visualization on the map.

Chapter 5

Discussion

This section discusses the main contributions, challenges, accomplishments, and future scope of our research.

Main contributions: We have successfully implemented a Deep Reinforcement Learning (DRL) based solution for automatically generating a (near) optimal policy for a realistic version of the CVRP. The model enacts an agent which can track the current state of the RL environment and hence decide the best action to take at each time step. It takes real-world raw graph features, such as location coordinates, customer demands, node distances, etc., as input and iteratively predicts the next node probability distribution, based on which an action decision is made. The critical component of our methodology is to include not just the node features but also the edge features during the encoding process. which ultimately leads to context-rich graph embedding representations. We also present a novel data generation process to create real-world graph instances that enable us to account for average traffic congestion levels in the city. This produces a more realistic prior distribution of the graph instances and hence the models trained on such datasets have higher applicability in social and industrial scenarios where combinatorial solutions are required. Extensive experiments conducted on these datasets prove that our proposed model outperforms the baseline ML method for almost all the problem sizes. Furthermore, our approach has relatively high scalability, but perhaps not as much as the OR-based solvers.

Let's talk data: The realistic routing problem is quite an interesting problem and in a way proposes a novel technique to augment the CVRP problem instances with domain knowledge about the setting. This augmentation strategy can be easily adapted to include other domain-specific information (such as delays due to weather, construction work, multiple-route options in

real-time, etc.) as per the complexity of the associated problem. Also, it is simple to reproduce, and can comfortably extend to new and more advanced use cases. As the power of our computational algorithms rises day by day, the need of having stringent performance evaluation criteria increases. This approach can thus be further extended to be developed as a benchmark generation technique for evaluating the performance of state-of-the-art ML and OR solvers. However, this avenue requires more extensive experimentation on the proposed method and was not possible in the limited scope of this work. In our implementation, we created multiple large-scale graph datasets (more than 200k graphs of varying sizes) with real coordinates and asymmetric distances - along with spatial and temporal delay adjustments - from the scratch.

Modelling challenges: While Transformer style multi-head attention was already proven to be an efficient feature extraction architecture to capture the graph context, our main motivation was to explicitly include the dynamic edge features so that the real-world changes in the distance (or transit cost) values are reflected on the graph embedding representations. To this extent, we were posed with the challenge of incorporating such delay-adjust edge features in the encoding process. Thankfully, after multiple iterations, we were able to develop a solution to encode the edge features during attention compatibility computation. The unstable training behavior of RL algorithms is widely known: setting up our custom training environment and stabilizing the training process was also one of the challenges faced by our team. However, we tried to follow an agile development and structured evaluation approach to efficiently develop our solution in the given time frame.

Experiments and performance: The experiment results, as presented in the previous section, indicate the superiority of the tour solutions generated by our proposed model as compared to the AM baseline model proposed in [29]. The average transit costs are significantly better than the baseline across all the graph sizes. However, this difference is slightly mitigated in *static* edge feature datasets where both approaches perform marginally the same for larger problems, and ours is better for smaller graph solutions. From our results, it is evident that edge features provide important information if carefully encoded in your policy network, although we didn't train till convergence, our model was able to find good and impressive solutions at 1000 step/epoch as shown in section4. In order to really know how powerful our model is, we might need to train to convergence and perhaps perform more rigorous fine-tuning procedure.

As mentioned above, one of the main challenges we faced during the

experimentation was generating and manipulating large problem instances while maintaining a reasonable execution time. For instance, our initial implementation took 1 hour, 3 hours, and 5 hours to generate 10000 instances of graph sizes 10, 20, and 30, respectively. However, we refined this strategy to generate these problem instances at about one-two hundredth of the initial execution time. With our current approach, it is possible to generate realistic graphs that have up to 500 customer locations. It is important to note that the storage capacity required grows exponentially as the graph size increases for the static and factorially for the dynamic datasets. Finally, the main challenge during the experimentation phase was to come up with optimized implementation strategies to reduce the training times and avoid out-of-memory errors. We employed multiple local and cloud computing instances (University Cluster, AWS Studio lab, and Paperspace's gradient) to manage our experiment protocol.

Future scope: We view our realistic implementation of the CVRP and its solution as a proof-of-concept that aims to develop a new class of realistic CVRPs. Such algorithms with immediate real-world applications have been widely gaining popularity in recent years, which can be further attributed to their increasing ability to solve these problems at scale. One of the future directions is thus to encourage research in such dynamic and realistic CVRP settings. Integrating such intelligent algorithms, which have high applicability /reproducibility for modern transportation systems, can provide transformational capabilities to enterprises while providing ease of mobility to the common man. However, building such systems will also require a rigorous training procedure based on the data priors which model real-life situations. Therefore, we also plan to benchmark and open-source our realistic dataset generation technique, which provides a more rigorous appropriate data distribution as well as a stricter evaluation benchmark to evaluate new algorithms. Our experiments have already proven that established ML-based methods are not performing even at par with the parallel savings heuristic for this realistic dataset, which poses a question about the performance and application of such models in real life. We want to encourage the researchers to adopt this scheme and hence foster research in Operation Research and ML methods for solving combinatorial optimization.

Chapter 6

Conclusion

In this student research project, we have successfully developed a DRL-based solution for solving a realistic version of the CVRP. The project kicked off with the motive of presenting proof-of-concept for solving real-life routing problems. In this direction, our primary contribution is an augmentation technique for the generation of large-scale realistic datasets of different graph sizes. We then propose a policy network architecture, inspired by the attention model in [29], that aims to capture the complexities and dynamism in real-world scenarios in order to learn a knowledge-rich graph embedding. The main idea was to propagate the effect of dynamic transportation systems in latent graph representations by explicitly incorporating the edge features during the encoding process. These graph contexts are then autoregressively decoded with the help of a multi-head attention module which selects the next node to visit at each time step until all customers are served. Extensive experiments conducted on realistically generated datasets proves that our method is more effective than the baseline approaches. Our research, specifically our data generation lays a strong foundation for future work in this domain.

References

[1] G. B. Dantzig and J. H. Ramser. "The Truck Dispatching Problem". In: *Management Science* 6.1 (1959), pp. 80–91. ISSN: 00251909, 15265501. URL: http://www.jstor.org/stable/2627477 (visited on 29/03/2023).

[2] Geoff Clarke and John W Wright. "Scheduling of vehicles from a central depot to a number of delivery points". In: *Operations research* 12.4 (1964), pp. 568–581.

[3] T. J. Gaskell. "Bases for Vehicle Fleet Scheduling". In: *Journal of the Operational Research Society* 18.3 (1967), pp. 281–295. DOI: 10.1057/jors.1967.44. eprint: https://doi.org/10.1057/jors.1967.44. URL: https://doi.org/10.1057/jors.1967.44.

[4] Billy E. Gillett and Leland R. Miller. "A Heuristic Algorithm for the Vehicle-Dispatch Problem". In: *Operations Research* 22.2 (1974), pp. 340–349. DOI: 10.1287/opre.22.2.340. eprint: https://doi.org/10.1287/opre.22.2.340. URL: https://doi.org/10.1287/opre.22.2.340.

[5] Richard H Mole and S. Jameson. "A Sequential Route-building Algorithm Employing a Generalised Savings Criterion". In: *Journal of the Operational Research Society* 27 (1976), pp. 503–511.

[6] David Ryan and Brian Foster. "An Integer Programming Approach to Scheduling". In: *Computer Scheduling of Public Transport* 1 (Jan. 1981), pp. 269–.

[7] H. Paessens. "The savings algorithm for the vehicle routing problem". In: *European Journal of Operational Research* 34.3 (1988), pp. 336–344. ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(88)90154-3. URL: https://www.sciencedirect.com/science/article/pii/0377221788901543.

[8] Manfred W. Padberg and Giovanni Rinaldi. "A branch-and-cut approach to a traveling salesman problem with side constraints". In: *Management Science* 35 (1989), pp. 1393–1412.

[9] George Kontoravdis and Jonathan F. Bard. "A GRASP for the Vehicle Routing Problem with Time Windows". In: *ORSA Journal on Computing* 7.1 (1995), pp. 10–23. DOI: 10.1287/ijoc.7.1.10. eprint: https://doi.org/10.1287/ijoc.7.1.10. URL: https://doi.org/10.1287/ijoc.7.1.10.

[10] Wen-Chyuan Chiang and Robert A. Russell. "Simulated annealing metaheuristics for the vehicle routing problem with time windows". In: *Annals of Operations Research* 63 (1996), pp. 3–27.

[11] Jacques Renaud, Fayez F. Boctor and Gilbert Laporte. "An Improved Petal Heuristic for the Vehicle Routeing Problem". In: *Journal of the Operational Research Society* 47 (1996), pp. 329–336.

[12] Gilbert Laporte, Michel Gendreau, Jean-Yves Potvin and Frédéric Semet. "Classical and modern heuristics for the vehicle routing problem". In: *International Transactions in Operational Research* 7.4 (2000), pp. 285–300. ISSN: 0969-6016. DOI: https://doi.org/10.1016/S0969-6016(00)00003-4. URL: https://www.sciencedirect.com/science/article/pii/S0969601600000034.

[13] Kenny Zhu. "A New Genetic Algorithm For VRPTW". In: (May 2000).

[14] Luca Maria Gambardella, Ric Taillard and Giovanni Agazzi. "MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows". In: (Aug. 2001).

[15] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8 (2004), pp. 229–256.

[16] Jean-François Cordeau, Gilbert Laporte, Martin Savelsbergh and Daniele Vigo. "Vehicle Routing". In: vol. 14. Jan. 2007, pp. 195–224.

[17] Jesus Gonzalez-Feliu. "Models and Methods for the City Logistics: The Two-Echelon Capacitated Vehicle Routing Problem". PhD thesis. May 2008.

[18] Roberto Baldacci, Aristide Mingozzi, Roberto Roberti and Roberto Wolfler Calvo. "An Exact Algorithm for the Two-Echelon Capacitated Vehicle Routing Problem". In: *Operations Research* 61.2 (2013), pp. 298–314. DOI: 10.1287/opre.1120.1153. eprint: https://doi.org/10.1287/opre.1120.1153. URL: https://doi.org/10.1287/opre.1120.1153.

[19] Ke-Lin Du and M. N. S. Swamy. "Introduction". In: *Search and Optimization by Metaheuristics: Techniques and Algorithms Inspired by Nature*. Cham: Springer International Publishing, 2016, pp. 1–28. ISBN: 978-3-319-41192-7. DOI: 10.1007/978-3-319-41192-7_1. URL: https://doi.org/10.1007/978-3-319-41192-7_1.

[20] Eva Volna and Martin Kotyrba. "Unconventional heuristics for vehicle routing problems". In: 9 (Jan. 2016), pp. 57–67.

[21] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi and Samy Bengio. *Neural Combinatorial Optimization with Reinforcement Learning*. 2017. arXiv: 1611.09940 [cs.AI].

[22] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal and Anand Subramanian. "New benchmark instances for the Capacitated Vehicle Routing Problem". en. In: *European Journal of Operational Research* 257.3 (Mar. 2017), pp. 845–858. ISSN: 03772217. DOI: 10.1016/j.ejor.2016.08.012. URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221716306270 (visited on 25/03/2023).

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[24] Oriol Vinyals, Meire Fortunato and Navdeep Jaitly. *Pointer Networks*. 2017. arXiv: 1506.03134 [stat.ML].

[25] Yoshua Bengio, Andrea Lodi and Antoine Prouvost. "Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon". In: *CoRR* abs/1811.06128 (2018). arXiv: 1811.06128. URL: http://arxiv.org/abs/1811.06128.

[26] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V. Snyder and Martin Takac. *Reinforcement Learning for Solving the Vehicle Routing Problem*. 2018. arXiv: 1802.04240 [cs.AI].

[27] Alex Nowak, Soledad Villar, Afonso S. Bandeira and Joan Bruna. *Revised Note on Learning Algorithms for Quadratic Assignment with Graph Neural Networks*. 2018. arXiv: 1706.07450 [stat.ML].

[28] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[29] Wouter Kool, Herke van Hoof and Max Welling. *Attention, Learn to Solve Routing Problems!* 2019. arXiv: 1803.08475 [stat.ML].

[30] Kenneth Sörensen, Florian Arnold and Daniel Palhazi Cuervo. "A critical analysis of the "improved Clarke and Wright savings algorithm"". In: *International Transactions in Operational Research* 26.1 (2019), pp. 54–63. DOI: https://doi.org/10.1111/itor.12443. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/itor.12443. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12443.

[31] Hanjun Dai, Bo Dai and Le Song. *Discriminative Embeddings of Latent Variable Models for Structured Data*. 2020. arXiv: 1603.05629 [cs.LG].

[32] Lu Duan, Yang Zhan, Haoyuan Hu, Yu Gong, Jiangwen Wei, Xiaodong Zhang and Yinghui Xu. "Efficiently Solving the Practical Vehicle Routing Problem: A Novel Joint Learning Approach". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*. KDD '20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, 3054–3063. ISBN: 9781450379984. DOI: 10.1145/3394486.3403356. URL: https://doi.org/10.1145/3394486.3403356.

[33] Luca Pasqualini and Maurizio Parton. *Pseudo Random Number Generation through Reinforcement Learning and Recurrent Neural Networks*. 2020. arXiv: 2011.02909 [cs.CR].

[34] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, Francois-Xavier Aubet, Laurent Callot and Tim Januschowski. *Deep Learning for Time Series Forecasting: Tutorial and Literature Survey*. 2022. arXiv: 2004.10240 [cs.LG].

[35] Jingwen Li, Yining Ma, Ruize Gao, Zhiguang Cao, Andrew Lim, Wen Song and Jie Zhang. "Deep Reinforcement Learning for Solving the Heterogeneous Capacitated Vehicle Routing Problem". In: *IEEE Transactions on Cybernetics* 52.12 (Dec. 2022), pp. 13572–13585. DOI: 10.1109/tcyb.2021.3111082.

[36] Ruibin Bai, Xinan Chen, Zhi-Long Chen, Tianxiang Cui, Shuhui Gong, Wentao He, Xiaoping Jiang, Huan Jin, Jiahuan Jin, Graham Kendall, Jiawei Li, Zheng Lu, Jianfeng Ren, Paul Weng, Ning Xue and Huayan Zhang. "Analytics and machine learning in vehicle routing research". In: *International Journal of Production Research* 61.1 (2023), pp. 4–30. DOI: 10.1080/00207543.2021.2013566. eprint: https://doi.

org/10.1080/00207543.2021.2013566. URL: https://doi.org/10.1080/00207543.2021.2013566.

[37] Jia Luo, Chaofeng Li, Shaopeng Shang, Yuhui Zheng and Yiwen Ju. "An Efficient Encoder-Decoder Network for the Capacitated Vehicle Routing Problem". In: (2023). DOI: http://dx.doi.org/10.2139/ssrn.4395195. URL: https://ssrn.com/abstract=4395195.

Chapter 7

Appendix

7.1 Training Hyperparameters and Detailed Evaluation Results

Detailed experiment results

Experiment	Best Reward	Mean Train Reward	Mean test reward	Test vs. Train Gap %
AM_D10	108.44	117.05	117.85	0.69%
AM_D20	226.95	265.11	253.34	-4.44%
AM_D30	321.70	390.69	385.07	-1.44%
AM_D50	515.72	637.28	631.94	-0.84%
Ours _D10	105.13	119.41	112.06	-6.15%
Ours _D20	219.00	240.49	236.88	-1.50%
Ours _D30	325.95	352.10	350.40	-0.48%
Ours_D50	522.22	655.74	652.42	-0.51%

Table 7.1: Final results for *dynamic* dataset. Values represent average tour lengths (in km) for test and train datasets. Gap % represents how better are test solutions as compared to the train.

Training scheme

7.2 The Training Plots and Comparison with Attention Model

The more training plots of mean test reward and loss on static and dynamic distance graphs of varying sizes 10, 20, 30, and 50 are plotted here.

Experiment	Best Reward	Mean Train Reward	Mean test reward	Test vs. Train Gap %
AM_S10	105.47	119.49	121.03	1.29%
AM_S20	180.95	202.30	209.52	3.57%
AM_S30	286.52	299.28	300.60	0.44%
AM_S50	461.17	575.37	546.44	-5.03%
Ours_S10	102.71	112.56	112.78	0.19%
Ours_S20	172.62	228.14	219.06	-3.98%
Ours_S30	253.31	296.84	301.41	1.54%
Ours_S50	455.57	571.91	556.37	-2.72%

Table 7.2: Final results for *static* dataset. Values represent average tour lengths (in km) for test and train datasets. Gap % represents how better are test solutions as compared to the train.

Model x Size	Ours	AM Baseline	OR solver
10	10.51	10.84	14.09
20	10.95	11.35	12.67
30	10.86	10.72	12.02
50	10.44	10.31	12.84
Combined	10.69	10.81	12.90

Table 7.3: Average transit cost per customer (in km) for various proposed implementations

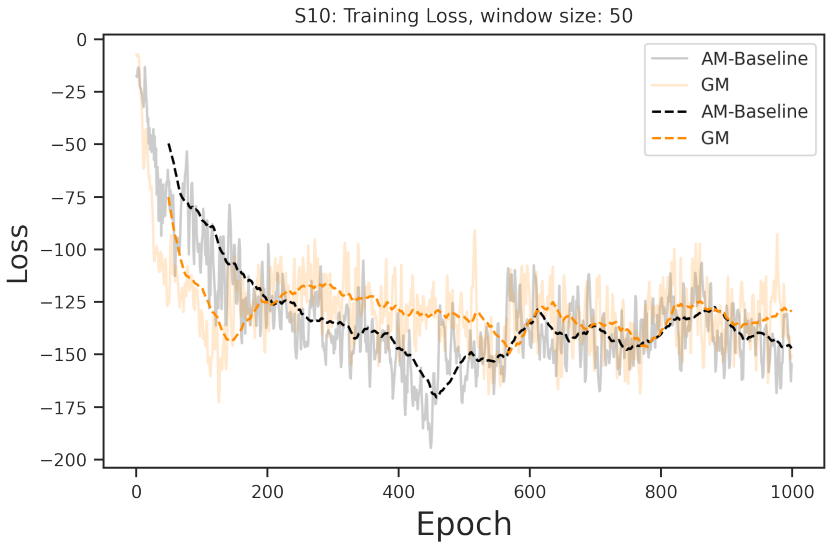


Figure 7.1: The training loss curve on static distance graphs of size 10.

Training parameter	Value(s)
Train size, Test size	(10,000, 10,000)
Graph sizes, N	(10, 20, 30, 40)
Max epochs trained	1,000
Learning rate, η	0.000022
LR scheduler eps	10^{-8}
Regularization decay, λ	0.86667
Batch size, B	32
Number of layers per attention stack, N_A	3
Attention heads, M	8
Initial embedding dimension, d_h	64
Hidden embedding dimension d_k	16
Tanh clipping parameter, C	10
Training buffer size	50,000
Steps per epoch	100*E
Episode per collect	100
Repeat per collect	1
Number of Environments, E	500

Table 7.4: Final Training hyperparameters for *Ours* (GM) model

Training parameter	Value(s)
Train size, Test size	(10,000, 10,000)
Graph sizes, N	(10, 20, 30, 40)
Max epochs trained	1,000
Learning rate, η	0.00005
LR scheduler eps	10^{-8}
Regularization decay, λ	0.005
Batch size, B	32
Number of layers per attention stack, N_A	2
Attention heads, M	8
Initial embedding dimension, d_h	64
Hidden embedding dimension d_k	16
Tanh clipping parameter, C	10
Training buffer size	50,000
Steps per epoch	100*E
Episode per collect	100
Repeat per collect	1
Number of Environments, E	500

Table 7.5: Final Training hyperparameters for *Attention* (AM) model

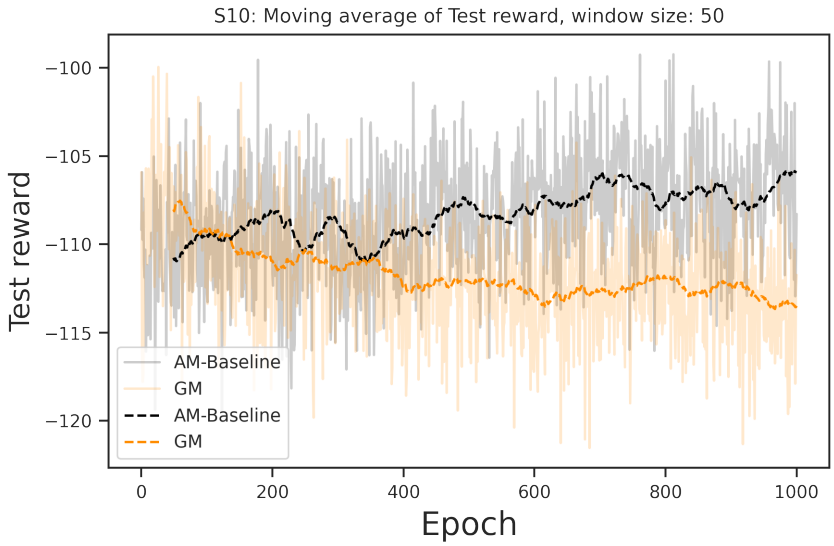


Figure 7.2: The mean test reward on static distance graphs of size 10.

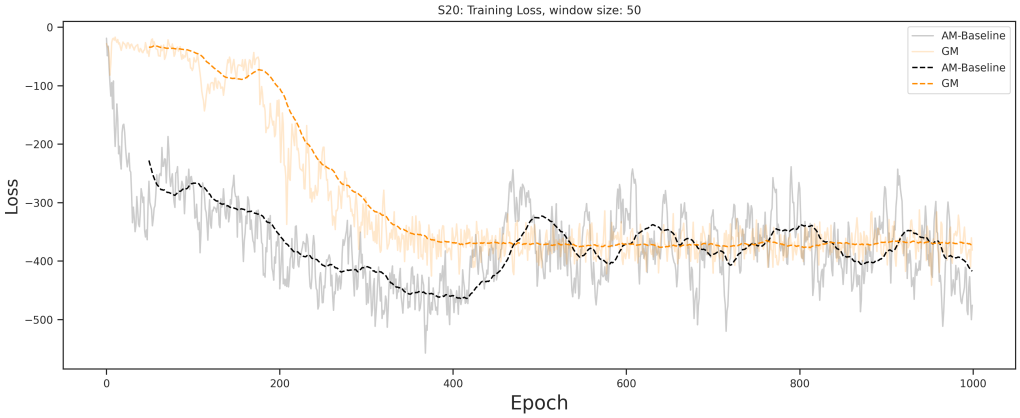


Figure 7.3: The training loss curve on static distance graphs of size 20.

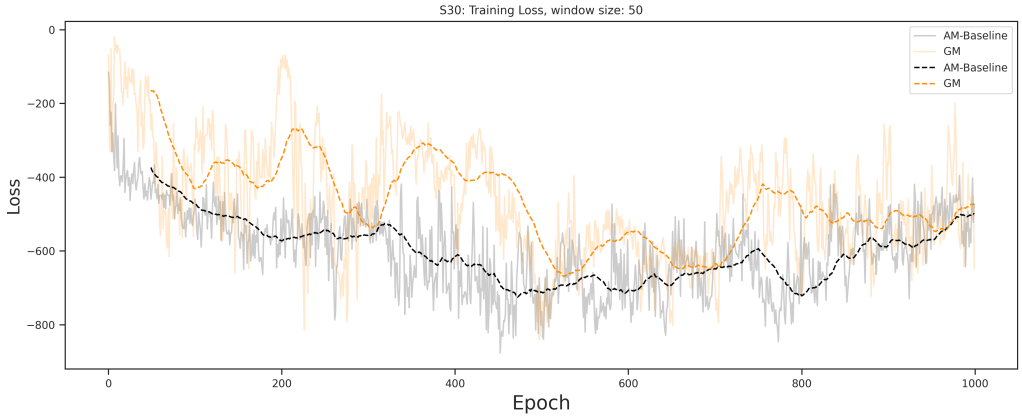


Figure 7.4: The training loss curve on static distance graphs of size 30.

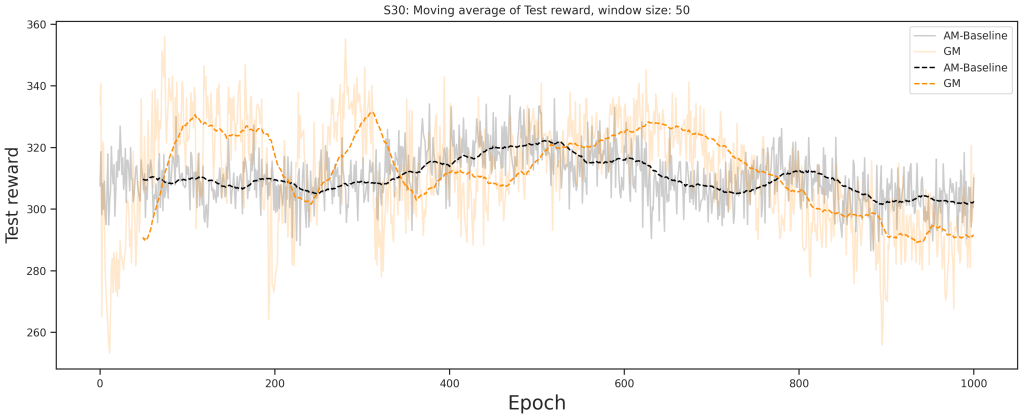


Figure 7.5: The mean test reward on static distance graphs of size 30.

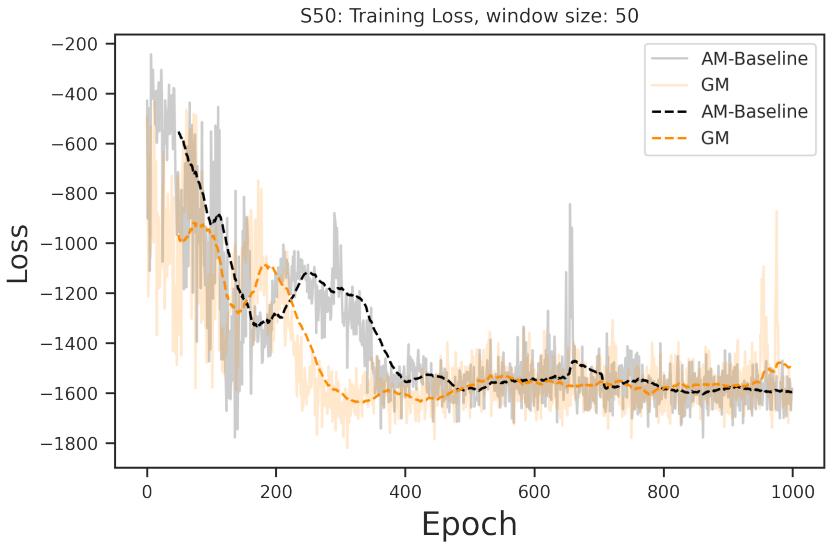


Figure 7.6: The training loss curve on static distance graphs of size 50.

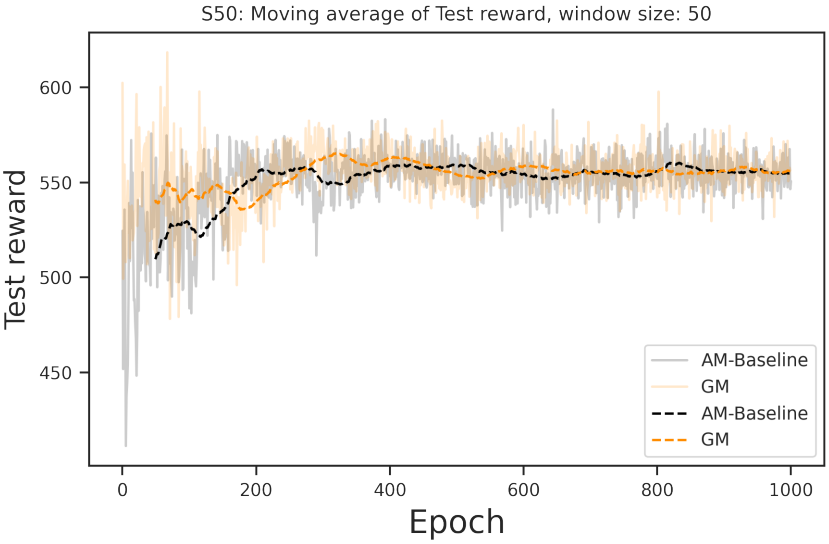


Figure 7.7: The mean test reward on static distance graphs of size 50.

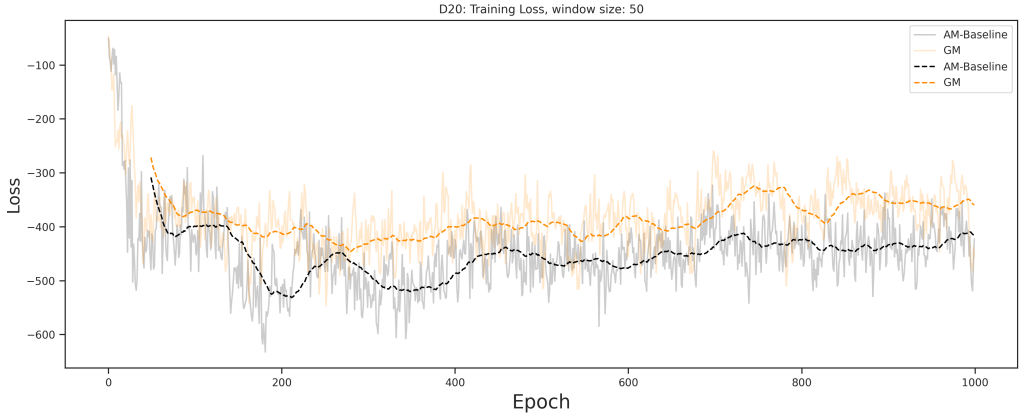


Figure 7.8: The training loss curve on dynamic distance graphs of size 20.

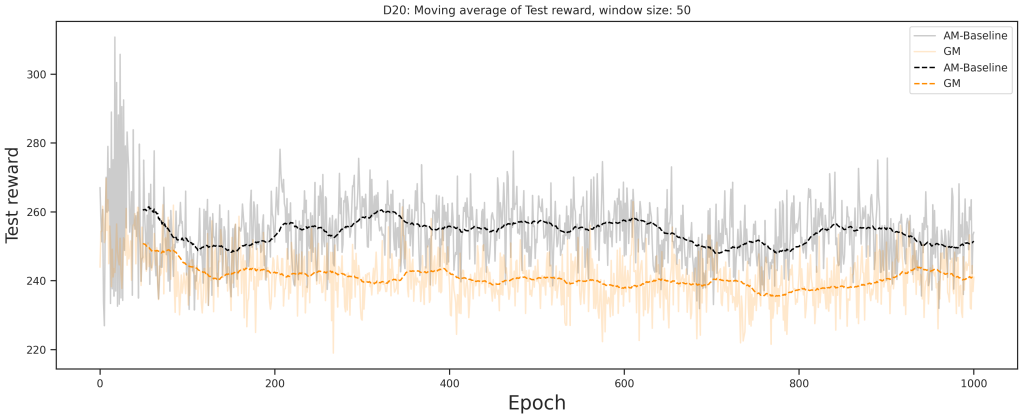


Figure 7.9: The mean test reward on dynamic distance graphs of size 20.

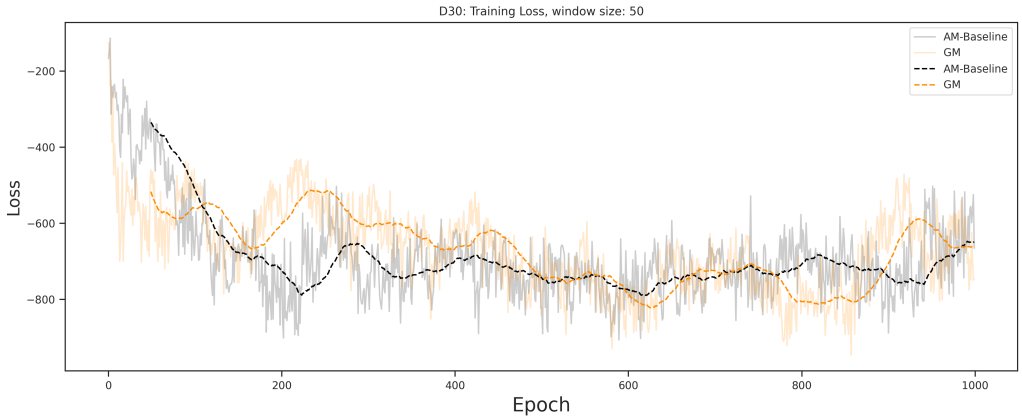


Figure 7.10: The training loss curve on dynamic distance graphs of size 30.

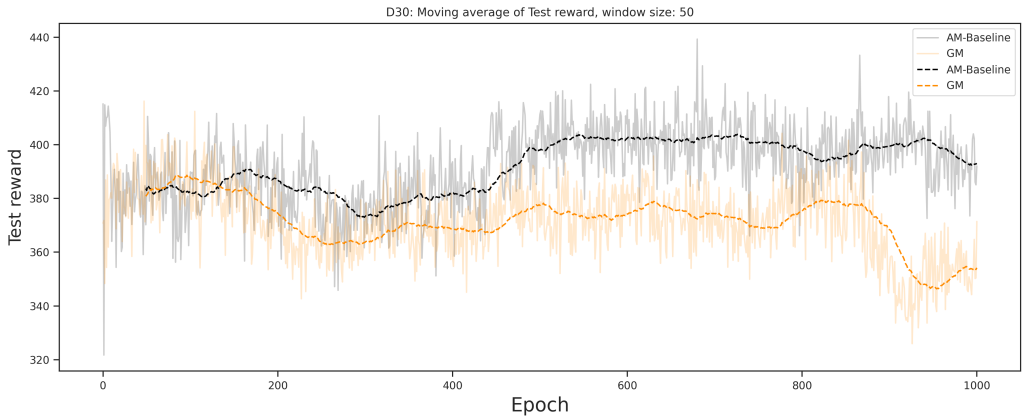


Figure 7.11: The mean test reward on dynamic distance graphs of size 30.